Escola d'Enginyeria de Telecomunicació i Aeroespacial de Castelldefels
UNIVERSITAT POLITÈCNICA DE CATALUNYA

TREBALL DE FI DE GRAU

TÍTOL DEL TFG: Desenvolupament d'un Launcher Android per a aplicacions e-health

TITULACIÓ: Grau en Enginyeria Telemàtica

AUTOR: Júlia Garrigós Sabaté

DIRECTOR: Roc Meseguer Pallarès

SUPERVISOR: Xavier Lagunas Calpe

DATA: 18 de setembre de 2013

Títol: Desenvolupament d'un Launcher Android per a aplicacions e-health

Autor: Júlia Garrigós Sabaté

Director: Roc Meseguer Pallarès

Supervisor: Xavier Lagunas Calpe

DATA: 18 de setembre de 2013

Resum

Aquest treball s'emmarca en el projecte CarismaTIC de l'Àrea de Salut i Dependència de la Fundació i2CAT. El CarismaTIC és una plataforma per a millorar l'atenció i la qualitat de vida dels pacients crònics i dels seus cuidadors. Ho fa a través de quatre serveis bàsics: comunicació, formació, seguiment i oci. Per tal d'oferir-los, es proporciona als pacients un tablet per a què puguin accedir a la plataforma des de casa.

El principal objectiu del projecte serà adaptar els tablets a les necessitats dels seus usuaris tenint en compte que és possible que no hagin tingut contacte abans amb dispositius d'aquest tipus i que, en la seva majoria, són gent gran. Per tant, s'ha d'aconseguir que els seus elements siguin el més intuïtius possible i s'ha de minimitzar la interacció amb l'usuari, automatitzant tot el que es pugui.

Una de les millors maneres de controlar el comportament d'un dispositiu és mitjançant l'escriptori. Això permet elegir quines aplicacions mostrar, com de grans són les icones, etc. Per això aquest projecte tracta del desenvolupament d'un escriptori per a Android (*launcher*). Aquest dóna accés als serveis del CarismaTIC a través d'aplicacions: una de videoconferència, una que mostra el pla de medicació, una que mostra informació sobre la malaltia del pacient, etc. Com que no tots els pacients tenen les mateixes necessitats, el seu metge pot triar quines aplicacions tindrà disponibles cada un, de manera que el Launcher ha de donar accés només a les aplicacions que el metge triï, i aquest ha de poder afegir-ne o eliminar-ne en qualsevol moment. Per això, el Launcher ha d'estar preparat per ser modificat de forma remota.

En l'actualitat no existeix cap sistema per a gent gran que estigui dissenyat pensant en les seves necessitats i que ofereixi gestió remota del contingut que es mostra a l'usuari. Aquest projecte pot aportar molt a l'àmbit de la teleassistència i la telerehabilitació, permetent a metges i pacients estar en contacte constant sense necessitat de desplaçaments. També permet al metge fer un seguiment més continu del seu pacient a través d'aplicacions que permetin fer exercicis, etc.

Title: TFC/PFC Model

Author: Júlia Garrigós Sabaté

Director: Roc Meseguer Pallarès

Supervisor: Xavier Lagunas Calpe

Date: September, 18th 2013

Overview

This project is framed within the CarismaTIC project of the i2CAT's Foundation Health & Dependence Area. CarismaTIC is a platform to improve the life quality of chronic patients and their caregivers, as well as the attention received by the patients. It achieves so through four basic services: communication, education, tracking and leisure. In order to offer these services to the patients, each of them is provided with a tablet, so they may access the platform from home

The main objective of this project is to adapt the tablets to the needs of their users, taking into account that they may have never been in contact with these kind of device and that most of them are elderly. Thus, the tablets have to be as intuitive as possible, being as automatized as possible in order to minimize user interaction. One of the best ways to control a device is through the desktop, what allows to choose what applications to show, the icon size, etc. For that reason, this project deals with the development of an Android desktop, or 'Launcher', which provides access to the CarismaTIC services through different apps: one for videoconferencing, one to show the treatment plan, one to show information about the patient's illness, etc. As not all the patients have the same needs, each patient's physician may choose what apps are available to the patient, so the Launcher of each patient only gives access to the apps chosen by his doctor, who may add or remove apps at any moment. Thus, the Launcher must be able to be remotely modified.

Nowadays, there is no system adapted to the elderly which is designed according to their needs and which offers remote management of the contents showed to the user. This project may contribute significantly to the teleassistance and telerehabilitation fields, allowing physicians and patients to be in constant contact but not requiring them to move. It also allows physicians to track their patients more closely through exercise apps, etc.

Thus, this project will allow to notably improve the life of patients, which, thanks to the tablet, will be able to learn about topics related to their illness, do exercises which will help them, engage in leisure activities, etc.

ÍNDEX

CAPÍTOL 1. INTRODUCCIÓ

Aquest projecte consisteix en el desenvolupament d'un *launcher* Android en l'àmbit de la salut electrònica (*e-health*). Un *launcher* és un tipus d'aplicació que es mostra quan s'inicia un dispositiu Android, permetent accedir a la resta d'aplicacions i s'executa cada vegada que es prem el botó d'inici (*home*). Es podria dir que el *launcher* és l'escriptori dels dispositius Android.

Aquest projecte s'ha realitzat en el marc de l'Àrea de Salut i Dependència de la Fundació i2CAT, que és un centre d'investigació i innovació que enfoca les seves activitats en el desenvolupament de la Internet del futur. El Launcher s'integrarà en el projecte CarismaTIC que té com objectiu crear un servei TIC que millori l'atenció i la qualitat de vida dels pacients crònics i dels seus cuidadors.

Hi ha dues motivacions bàsiques que porten a la realització d'aquest projecte. Per una banda la necessitat de crear aplicacions amigables i simples per a gent gran, que té necessitats molt concretes d'usabilitat; per exemple, necessiten que els botons siguin grans per veure clarament què estan fent. En desenvolupar una aplicació Android per a aquest tipus d'usuaris sorgeix un problema: es pot controlar tot el contingut i fer-lo usable dins de l'aplicació; però no es pot modificar la icona que apareix a l'escriptori del dispositiu perquè aquesta depèn del Launcher. Un dels objectius d'aquest projecte és poder personalitzar l'escriptori dels dispositius per fer-lo el més usable possible.

Per altra banda, existeix la necessitat d'una aplicació multifuncional modificable remotament, que permeti triar les funcionalitats que tindrà disponible cada usuari en funció de les seves necessitats. En aquest projecte hi haurà diversos tipus de d'usuaris i es requereix que el seu metge pugui triar el contingut que apareixerà al dispositiu de cada un. Per aconseguir-ho, el servidor tindrà un llistat d'aplicacions d'entre les quals el metge triarà les més adequades per a cada pacient. D'aquesta manera, els dispositius només mostraran les aplicacions que els han estat assignades. Per exemple, per un pacient amb Alzheimer, el metge podria triar l'aplicació de videoconferència, per fer un seguiment periòdic del seu estat mitjançant les videotrucades, i també un joc perquè realitzi exercicis de memòria. En canvi, per a un pacient amb problemes respiratoris, podria elegir l'aplicació de videoconferència i una aplicació amb informació sobre malalties respiratòries.

El propòsit d'aquest projecte és dissenyar i implementar una aplicació Android que actuï com a *Launcher*. Complint certs requeriments per facilitar l'ús de l'aplicació a gent gran i persones discapacitades; permetent la modificació remota dels seus continguts.

Aquest capítol, descriu el context en què es realitza el projecte, les motivacions que han portat a la seva realització i els objectius que es volen complir.

1.1. Marc

El desenvolupament del projecte es durà a terme a l'Àrea de Salut i Dependència de la Fundació i2CAT. En aquest departament es realitzen projectes relacionats amb la telemedicina, la telerehabilitació, la teleassistència, etc. El Launcher s'integra en el CarismaTIC, un projecte, l'objectiu del és crear un servei TIC que permeti millorar l'atenció i la qualitat de vida de pacients crònics i dels seus cuidadors. El CarismaTIC pretén comunicar metges, assistents socials i beneficiaris a través d'internet. Per aconseguir-ho es formaran *grups de suport* compostos per: N beneficiaris, N metges i N col·laboradors (assistents socials o personal mèdic de suport que realitza tasques de seguiment dels malalts).

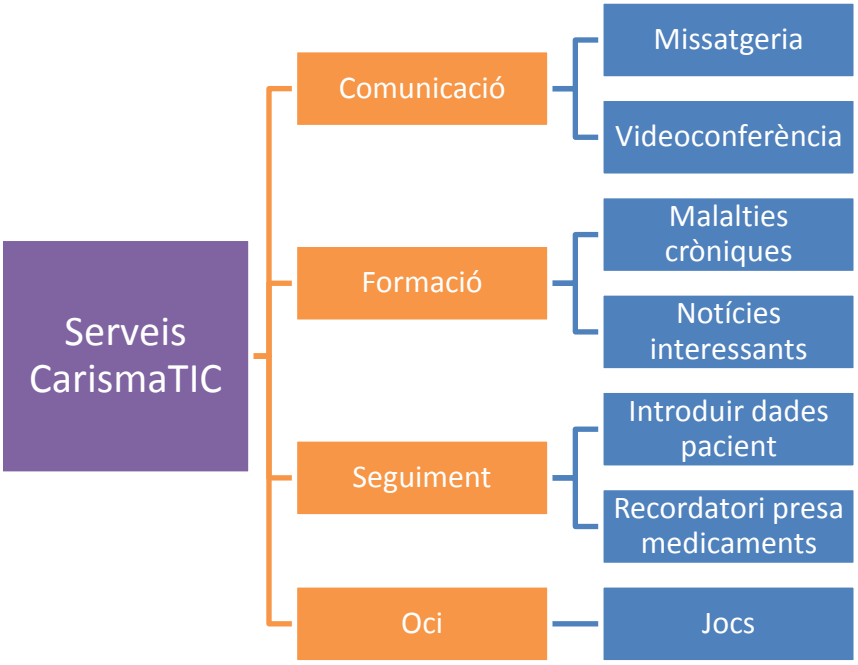

Fig. 1 Esquema dels serveis que ha de donar el CarismaTIC i exemples d'algunes de les funcionalitats que han d'oferir.

S'oferiran diferents serveis als usuaris, que es poden dividir en quatre grups (veure la Fig. 1):

- **Comunicació**: una aplicació de missatgeria i una de videoconferència que permetran la comunicació entre els membres del grup de suport.
- **Formació**: diferents aplicacions que donaran als beneficiaris informació sobre la seva malaltia i sobre altres temes d'interès.
- **Seguiment**: una aplicació que recordarà al pacient els moments de presa de medicaments, quan té una cita de videoconferència amb el seu metge, etc. També disposarà d'una aplicació a través de la qual el col·laborador introduirà les dades obtingudes en els seguiments que realitzarà a casa del beneficiari (antecedents mèdics, dades mèdiques,

etc.). Aquestes dades podran ser consultades a través de la web pel metge.
- **Oci**: constarà de diferents jocs que permetran fer exercicis que requereixen memòria, lògica, etc.

La comunicació entre els diferents usuaris del CarismaTIC es durà a terme a través d'internet. Per a accedir a la plataforma, els metges empraran navegadors web, els beneficiaris faran servir *tablets* Android i els col·laboradors accediran a través dels dispositius dels beneficiaris per introduir les seves dades, i a través del web per a fer la resta de gestions. Aquest TFG tractarà el desenvolupament de l'aplicació que permetrà als beneficiaris accedir al CarismaTIC a través del *tablet*: el Launcher. El Launcher donarà accés als serveis del CarismaTIC a través de diferents aplicacions. Aquestes, poden ser desenvolupades específicament per al CarismaTIC o poden ser aplicacions extretes del Play Store. Com que cada pacient té unes necessitats diferents, el Launcher serà personalitzable. El metge triarà les aplicacions que ha de veure cada usuari i podrà modificar-les remotament quan ho cregui convenient

1.2. Motivació i propòsit del projecte

A l'Àrea de Salut i Dependència d'i2CAT s'han realitzat diferents projectes de teleassistència que tenien gent gran com usuaris finals. En la majoria d'aquests projectes l'usuari havia d'accedir a la plataforma mitjançant un *tablet* i van sorgir problemes relacionats amb l'ús d'aquest dispositiu. Els usuaris no estaven habituats a l'ús d'aparells d'aquest tipus i a més, aquests no estaven dissenyats de manera que fos fàcil que s'hi adaptessin.

El principal problema que sorgia, era que l'usuari havia d'executar l'aplicació des de l'escriptori del seu *tablet* (que és diferent en cada model) i el desenvolupador no podia assegurar que aquest fos usable ja que no tenia permisos per modificar-lo. Tal com es pot observar a la Fig. 2, que mostra l'escriptori del *tablet* Samsung Galaxy Tab 2, les icones són petites i poc intuïtives per a persones poc habituades a l'ús d'aquests aparells.

Per tant, per complir els objectius del CarismaTIC, a més que l'aplicació s'adapti a les necessitats de cada beneficiari oferint-li només els serveis triats pel seu metge, cal que el *tablet* s'adapti a ell i sigui amigable. A continuació es plantegen tres possibles solucions avaluant els seus avantatges i inconvenients.

La primera solució consisteix en crear una aplicació "normal" que permeti mostrar certs continguts en funció de l'usuari que hi accedeixi. Aquest tipus d'aplicació té l'inconvenient que caldria que l'usuari executés l'aplicació des de l'escriptori del seu *tablet* que, tal com s'ha vist, no és el més adequat.

Fig. 2 Exemple de Launcher Android en el que es poden veure les icones que porten a les diferents aplicacions.

La segona solució pretén resoldre el problema de la primera fent que l'aplicació estigui sempre activa als dispositius. Això es pot aconseguir convertint-la en el Launcher del *tablet*; de manera que quan s'encengui l'aparell sigui el primer en aparèixer i que quan es premi el botó *home* torni a executar-se. D'aquesta manera es pot assegurar que l'usuari no tindrà problemes d'accés a l'aplicació i que res quedarà fora del control dels desenvolupadors. Tanmateix, té un inconvenient: es limiten molt les possibilitats del dispositiu, en el qual només es pot fer servir aquesta aplicació ja que no es dóna opció d'obrir-ne una altra. Així, si altres membres de la família del beneficiari volen fer servir el *tablet,* no podrien fer-ho.

Per últim, la tercera solució consisteix en convertir l'aplicació anterior en un autèntic Launcher, és a dir, una aplicació que faci la funció d'escriptori. Aquesta donaria accés a les aplicacions instal·lades al dispositiu i alhora permetria la personalització que exigeixen les necessitats dels usuaris. Aquest tipus d'aplicació aporta els següents avantatges:

- Permet controlar completament la interfície d'accés de l'usuari, de manera que es pot adaptar a les seves necessitats.
- Permet afegir accessos directes a aplicacions de tercers, la qual cosa facilita l'ampliació de l'oferta d'aplicacions de la plataforma.
- Com que es tractaria d'un escriptori, es pot dividir la plataforma en diferents mòduls on cada un ofereixi un servei diferent, així doncs, només s'instal·laran al dispositiu els mòduls que es consideri que són necessaris per l'usuari.

- Es facilita l'adaptació per a altres usos. Canviant les aplicacions, el sistema podria adaptar-se a públics molt diferents com nens petits o persones discapacitades.

Un altre problema molt comú que sorgia amb l'ús de *tablets* era la recepció de notificacions (en rebre trucades, missatges, recordatoris de presa de medicaments, etc.). Les notificacions generades pel sistema operatiu es mostren en un espai molt reduït i amb una mida de lletra petita. A més, el seu so d'alerta no es pot configurar per sonar fins que l'usuari rebi la notificació. Per aquest motiu el Launcher haurà de comptar amb un sistema que asseguri que l'usuari rep les notificacions i que aquestes són el més intuïtives possible.

Totes aquestes motivacions porten a la realització d'aquest projecte: un Launcher Android personalitzable remotament.

1.3. Arquitectura del CarismaTIC

L'objectiu del CarismaTIC és crear una plataforma formada per diferents mòduls on cada un d'ells ofereixi un servei independent. D'aquesta manera es deixa oberta la plataforma a possibles ampliacions. Per afegir una funcionalitat només caldrà afegir un mòdul nou i configurar remotament els *tablets* perquè la mostrin als usuaris.

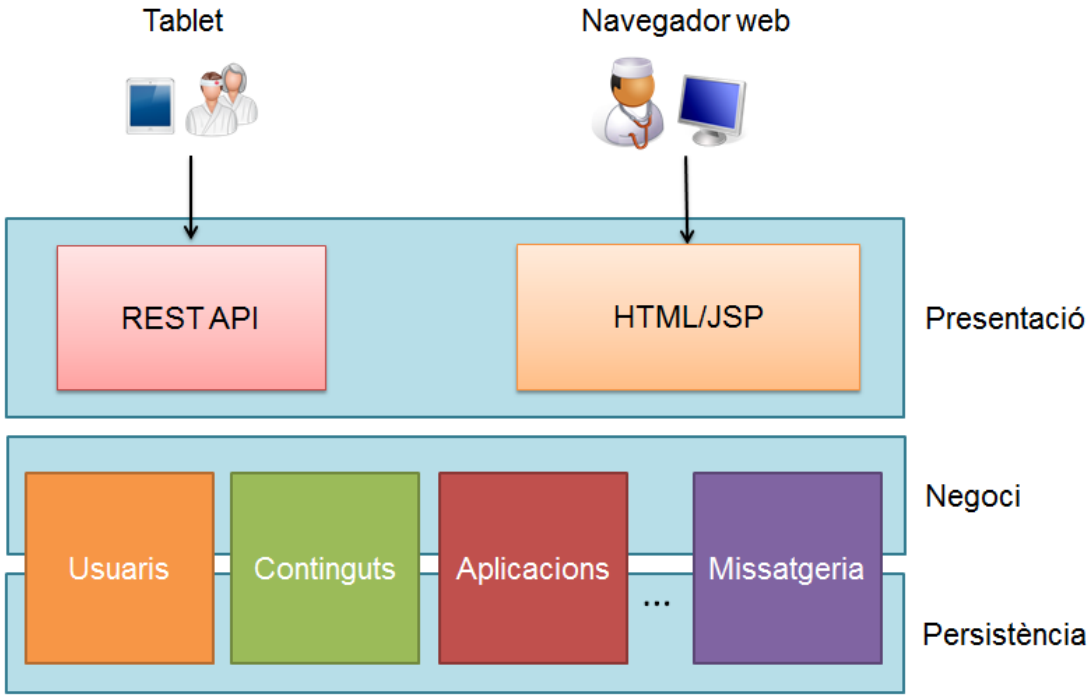

Fig. 3 Arquitectura del CarismaTIC.

A la Fig. 3 es pot observar l'arquitectura del CarismaTIC separada en tres capes: presentació, negoci i persistència. La capa de persistència és

l'encarregada d'accedir a la base de dades i d'assegurar que les dades es mantindran tot i que el servidor es reinicïi. La capa de negoci afegeix intel·ligència a la plataforma i la capa de presentació permet als usuaris accedir als serveis. La presentació la duran a terme dos components: una API REST que permetrà als dispositius mòbils accedir al servidor i JSPs que donaran accés als navegadors web.

Alguns dels mòduls que formaran part del CarismaTIC són els següents: mòdul d'usuaris, encarregat de la gestió dels usuaris, grups de suport, etc. mòdul de continguts, encarregat dels continguts de formació pels beneficiaris; mòdul d'aplicacions que permetrà al metge triar quines aplicacions tindrà disponibles cada pacient, etc.

1.4. Objectius del TFG

Aquest projecte es centra en el desenvolupament de dos dels components del CarismaTIC: el Launcher del dispositiu mòbil i l'API REST a través de la qual accedirà al servidor. El desenvolupament de la resta d'aplicacions mòbils, juntament amb l'aplicació web i la lògica i persistència del servidor, no seran explicats en aquest document.

Els objectius que es volen complir són:

- **Launcher Android**:
  - Dissenyar-lo de manera que sigui usable i que faciliti al màxim la interacció amb l'usuari.
  - Crear un entorn que permeti ser modificat remotament per així assegurar que el metge en tot moment podrà realitzar els canvis que cregui convenients.
  - No bloquejar les opcions del dispositiu de manera que els familiars de l'usuari també puguin fer servir el *tablet*.

- **API REST**:
  - Dissenyar un protocol de comunicació
  - Dissenyar una interfície d'accés al servidor que proporcioni seguretat i confidencialitat als usuaris.

CAPÍTOL 2. ANÀLISI

En aquest apartat s'analitza el sistema a dissenyar i implementar. S'identifiquen els diferents actors del CarismaTIC i se n'expliquen els casos d'ús. Basant-se amb aquesta informació, es descriuen els requeriments que ha de complir el projecte.

2.1. Casos d'ús

En aquest apartat s'analitzaran els actors del CarismaTIC i les històries d'usuari que han de complir. Els diagrames de casos d'us dels actors es poden trobar a l'ANNEX A. El CarismaTIC té quatre tipus d'usuaris molt diferenciats:

- **Administrador**: serà la persona encarregada de les tasques de gestió de la plataforma, el seu objectiu principal serà registrar totes les dades necessàries al sistema perquè el metge i el col·laborador hi puguin accedir i realitzar les seves tasques. Aquest rol el durà a terme personal de gestió que estigui al servei del metge: secretaris, infermers, etc.
- **Metge**: els metges faran servir els serveis de la plataforma per comunicar-se, tractar, formar, etc. als beneficiaris.
- **Col·laborador**: aquest rol el duran a terme treballadors socials, personal mèdic de suport, etc. que acudiran a les residències dels beneficiaris. Accediran a la plataforma mitjançant el *tablet* dels pacients per introduir-hi dades d'aquests; i a través del web per visualitzar les dades, comunicar-se amb els beneficiaris, etc.
- **Beneficiari**: es tracta del pacient crònic que es beneficiarà dels serveis que ofereix la plataforma. Pot accedir-hi directament emprant un *tablet* o pot fer-ho en lloc seu la persona que el cuida.

2.1.1. Model de casos d'ús de l'administrador

A la Fig. 37 es poden observar els casos d'ús de l'administrador. Podrà registrar, modificar i eliminar aplicacions. També, s'encarregarà del registre, l'assignació de rols, la modificació i l'eliminació d'usuaris. A més, haurà de crear i organitzar els grups de suport.

Haurà d'introduir al sistema els continguts de formació que puguin ser interessants per als beneficiaris i etiquetar-los de manera que sigui fàcil per al metge assignar-los als pacients que cregui convenients. Finalment, l'administrador haurà de poder accedir al registre (*Log*) del servidor en el qual se l'informarà de tots els accessos i canvis que s'han fet a la plataforma.

2.1.2. Model de casos d'ús del metge

Com que l'administrador introduirà a la plataforma les dades dels continguts i les aplicacions, el metge només els haurà d'assignar als beneficiaris que cregui convenient.

Tal com es mostra a la Fig. 38, el metge podrà comunicar-se amb els beneficiaris i els col·laboradors a través de la plataforma mitjançant missatgeria i videoconferència (només amb beneficiaris). Els missatges podran ser de tres tipus: text, àudio i vídeo. A més, podrà organitzar cites i afegir recordatoris a l'agenda dels beneficiaris.

Per altra banda, podrà afegir i consultar dades mèdiques i antecedents dels beneficiaris. Finalment, també podrà consultar, crear i modificar els plans de medicació dels beneficiaris.

2.1.3. Model de casos d'ús del col·laborador

A la Fig. 39 es mostren els casos d'ús del col·laborador, que podrà comunicar-se amb beneficiaris i metges a través de missatgeria i podrà fer videoconferències amb els beneficiaris. També podrà afegir cites a l'agenda dels beneficiaris remotament.

Per altra banda, quan vagi a la residència d'un beneficiari podrà consultar a través del *tablet* d'aquest el seu pla de medicació i antecedents mèdics. A més, si realitza mesures de dades mèdiques com pressió sanguínia, nivell de glucosa en sang, etc. podrà introduir els resultats a través del *tablet* de manera que el metge els pugui consultar.

2.1.4. Model de casos d'ús del beneficiari

Tal com es pot observar a la Fig. 40, els beneficiaris podran enviar missatges a metges i col·laboradors però no els podran trucar. Per mantenir-hi videoconferències haurà d'esperar a ser trucat. En canvi, si que podrà trucar (i enviar missatges) als beneficiaris del seu grup de suport.

El beneficiari també haurà de poder afegir cites a la seva agenda (hi observar-hi les cites que els metges i col·laboradors hi hagin afegit remotament) i fer servir el *tablet* per activitats d'oci. A més, aquest s'ha de poder fer servir com qualsevol altre *tablet*, és a dir, no pot estar limitat pel fet d'incorporar el CarismaTIC. El més probable és que el beneficiari només faci servir les funcions del CarismaTIC ja que no tindrà prou coneixements per emprar-ne d'altres, però els seus cuidadors i familiars podran aprofitar totes les opcions que ofereix l'aparell.

El beneficiari tindrà accés a continguts de formació assignats pel seu metge que contindran informació de la seva malaltia, notícies interessants, etc. També

podrà consultar el seu pla de medicació i, com el col·laborador, podrà afegir dades mèdiques a la plataforma perquè el metge pugui consultar-les

2.2. Requeriments del TFG

Els casos d'ús del beneficiari es compliran mitjançant aplicacions integrades al Launcher, com per exemple la de videoconferència, la de missatgeria, la de medicació, etc. Aquestes aplicacions no formen part d'aquest projecte i no seran explicades en aquest document.

El projecte pretén crear un entorn en el qual qualsevol aplicació es pugui integrar i funcionar de la manera més fàcil possible. El Launcher permetrà a les altres aplicacions instal·lar-se, accedir a les credencials de l'usuari, rebre notificacions i ser actualitzades remotament. Pel que fa a les actualitzacions, en general són gestionades pel Play Store de Google però en aquest projecte és possible que les aplicacions no estiguin al Play Store i, per tant, s'haurà de buscar una manera alternativa d'actualitzar-les.

Hi ha dos tipus de requeriments: els funcionals, que defineixen el comportament que ha de tenir l'aplicació; i els no funcionals, que no aporten cap funcionalitat a l'usuari però són necessaris pel correcte funcionament del sistema.

2.2.1. Requeriments funcionals

Els requeriments funcionals que haurà de complir l'aplicació -tenint present que les aplicacions mòbils que s'han de desenvolupar seran emprades per gent gran i per tant es pretén facilitar l'ús del dispositiu- són els següents:

- Comportar-se com el Launcher del dispositiu substituint el que aquest portés de sèrie.
- Demanar nom d'usuari i contrasenya a l'usuari per poder identificar-lo, però només una vegada. Després, ha de facilitar que totes les aplicacions facin servir aquestes credencials per accedir al servidor, de manera completament transparent a l'usuari.
- Accedir al servidor i descarregar el seu llistat d'aplicacions que haurà estat definit pel metge corresponent. Aquest llistat contindrà les aplicacions que haurà de mostrar a l'escriptori del dispositiu.
- Permetre organitzar les aplicacions en funció del servei del CarismaTIC al qual pertanyen.
- Encarregar-se de la gestió de les notificacions de tramesa automàtica (*push*) que s'empraran per notificar canvis en el llistat d'aplicacions, actualitzacions d'aplicacions, trucades entrants, missatges entrants, etc.
- Mostrar a l'usuari informació general: nivell de la bateria, connectivitat a internet, data i hora.
- Si alguna de les aplicacions del llistat no està instal·lada al dispositiu, el Launcher ha de descarregar-la i instal·lar-la.
- Permetre actualitzar les aplicacions, tan si estan al Play Store com si no.

- En el cas de les aplicacions que no es troben al Play Store, si tenen una actualització disponible, s'ha de donar l'opció d'instal·lar-la però també s'ha de permetre obrir l'aplicació sense estar obligat a instal·lar l'actualització.
- Un cop el dispositiu ha carregat el llistat d'aplicacions ha d'estar preparat per actualitzar-lo en qualsevol moment si es realitza algun canvi de forma remota. Per tant, el servidor haurà de notificar al dispositiu que s'ha produït un canvi en el llistat d'aplicacions, i aquest haurà de descarregar-lo una altra vegada i mostrar els canvis a l'usuari.
- No s'han de bloquejar funcionalitats del dispositiu.
- S'ha de permetre eliminar les dades confidencials del dispositiu perquè és possible que durant el procés d'avaluació del Launcher, un mateix *tablet* sigui emprat primer per un usuari i després per un altre.
- Mostrar en un lloc significatiu els logos dels patrocinadors i d'i2CAT.

2.2.2. Requeriments no funcionals

En la mateixa línia que a l'apartat anterior, en definir els requeriments no funcionals s'ha tingut en compte el tipus d'usuari concret que farà servir l'aplicació i les seves limitacions. A més, com que les dades que es transmetran en molts casos seran de caràcter mèdic, s'ha de garantir la confidencialitat i seguretat de les dades en totes les comunicacions. A continuació es llisten els requeriments no funcionals que haurà de complir el Launcher:

- **Seguretat**: les comunicacions han de ser segures. Tota connexió amb el servidor ha d'anar xifrada i ha de ser autenticada.
- **Usabilitat**:
  - Tots els botons han de ser clars i anar acompanyats de text
  - Les opcions han de ser el més intuïtives possible
  - S'han de mostrar missatges d'error sempre que n'hi hagi algun
  - S'han de minimitzar les accions que realitza l'usuari automatitzant-ho tot el màxim possible.
- **Estabilitat**: ha de considerar tots els errors possibles i no penjar-se.
- **Portabilitat**: ser compatible amb dispositius amb pantalles de 7 i 10 polzades amb Android 4.0.3 o superior.
- **Multi-idioma**: estar disponible en català i en castellà i tenir la possibilitat d'afegir un altre idioma fàcilment.
- **Disponibilitat**: s'ha de permetre utilitzar el Launcher encara que l'usuari no disposi de connexió a internet (sempre que les aplicacions així ho permetin, per exemple, l'aplicació de videoconferència no funcionarà sense connexió mentre que un joc de memòria sí que ho farà).

CAPÍTOL 3. DISSENY I IMPLEMENTACIÓ DEL LAUNCHER

En aquest capítol es descriu el disseny i la implementació del Launcher. Es comença explicant com s'aconsegueix que l'aplicació es comporti com un Launcher, i a continuació es descriu el seu disseny visual, es detalla el model de dades que representarà les aplicacions i, per acabar, es descriuen els components que formen l'aplicació.

A l'ANNEX B es detallen les tecnologies que s'han fet servir per desenvolupar aquest projecte.

3.1. Aplicació tipus Launcher

El més important d'aquesta aplicació és que es comporti com l'escriptori del dispositiu. Per a aconseguir-ho s'ha d'especificar que la seva *Activity* principal (*MainActivity*) és un Launcher.

```
<activity
          android:name="net.i2cat.csade.launcher.activities.MainActivity"
          android:configChanges="orientation|screenSize"
          android:label="@string/title_activity_main"
          android:launchMode="singleTask"
          android:theme="@style/CustomTheme" >
          <intent-filter>
              <action android:name="android.intent.action.MAIN" />
              <category android:name="android.intent.category.HOME" />
              <category android:name="android.intent.category.DEFAULT" />
              <category android:name="android.intent.category.LAUNCHER" />
          </intent-filter>
      </activity>
```

Fig. 4 Declaració del l'*Activity MainActivity* del Launcher a *l'AndroidManifest.xml.*

Quan es defineix una *Activity* a l'*AndroidManifest.xml* se li poden afegir *Intent Filters* que defineixen les seves característiques. Per a especificar que una aplicació s'ha de comportar com un *launcher* s'ha d'afegir l'*Intent Filter android.intent.category.HOME* a la seva *Activity* principal. Un altre exemple d'*Intent Filters* serien els que fan que l'aplicació aparegui al llistat d'aplicacions del dispositiu: *android.intent.action.MAIN* i *android.intent.category.LAUNCHER.* A la Fig. 4 es mostra la definició de *l'Activity* principal del Launcher.

Quan en un *tablet* s'instal·la un *launcher* nou (el dispositiu sempre en porta un de sèrie) i es prem el botó *Home,* apareix en pantalla un menú que deixa triar quin dels *launchers* (el de sèrie o el nou) es vol executar. Aquest menú també permet establir un dels *launchers* com predeterminat, de manera que s'executi sempre que es premi aquest botó.

3.2. Disseny visual

El Launcher haurà de crear un entorn usable que permeti a l'usuari accedir a les aplicacions assignades pel seu metge. Per altra banda, no pot bloquejar les opcions del *tablet* per si els familiars i cuidadors del beneficiari volen fer-lo servir. Per tant, haurà de donar accés a la resta d'aplicacions que té el dispositiu instal·lades.

A més, ha de permetre reconfigurar els paràmetres que no siguin controlables de forma remota. I, com que és possible que en l'avaluació del CarismaTIC un mateix *tablet* primer sigui emprat per un usuari i després sigui cedit a un altre, és important que permeti eliminar les dades confidencials del dispositiu.

Tots aquests requeriments han estat considerats en el disseny del Launcher, que constarà de quatre vistes que es detallen als apartats següents.

A l'ANNEX C s'explica com el Launcher suporta canvis d'orientació i d'idioma i com s'aconsegueix que sigui compatible amb dispositius amb pantalles de diferents mides.

3.2.1. Vista d'Inici de Sessió

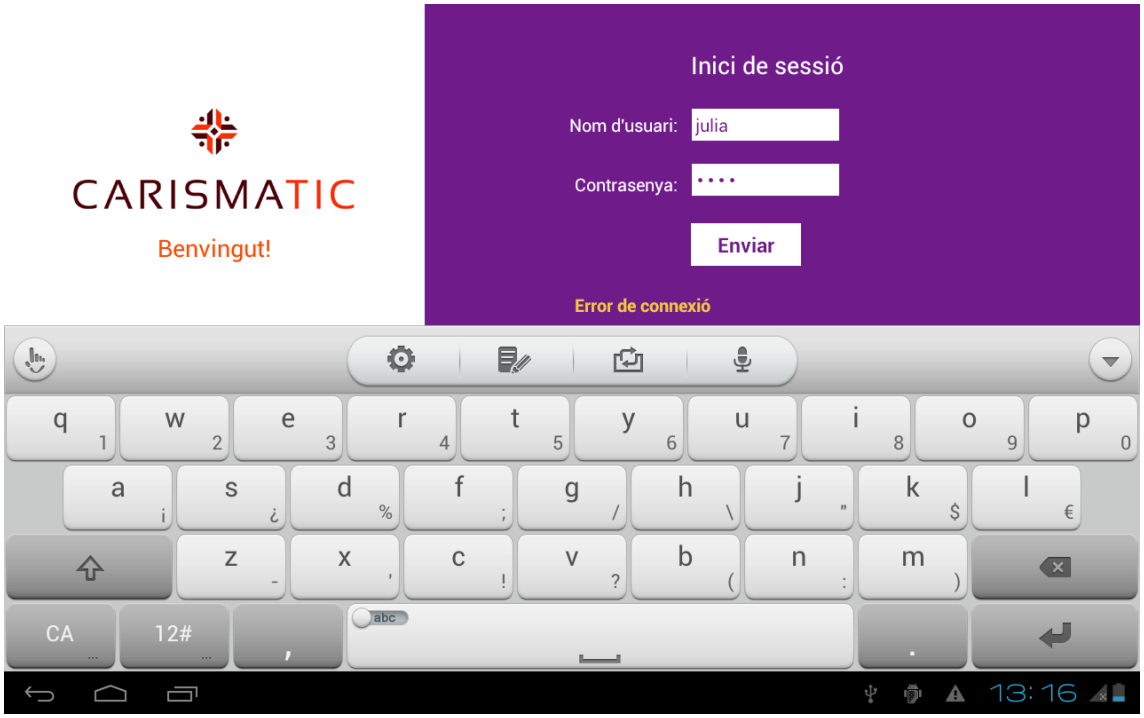

Fig. 5 Captura de la *Vista d'Inici de Sessió*. En aquest exemple l'usuari ha introduït les seves credencials, però al enviar-les s'ha produït un error de connexió.

En aquesta vista és dóna la benvinguda a l'usuari i se li permet iniciar sessió mitjançant el seu nom d'usuari i la seva contrasenya. Si inicia sessió correctament s'obre la *Vista Principal*, si no, se li mostra l'error corresponent.

A la Fig. 5 es mostra la *Vista d'Inici de Sessió*, que ha estat dissenyada considerant que en tot moment es mostrarà el teclat i per això no té cap element a la part inferior de la pantalla.

3.2.2. Vista Principal

La *Vista Principal* ha de donar accés a les aplicacions del CarismaTIC que hagin estat elegides per al beneficiari. A més, ha de permetre ordenar-les en les quatre categories que representen els quatre serveis del CarismaTIC: formació, seguiment, comunicació i oci. També ha de donar accés a la resta d'opcions del dispositiu, com el llistat d'aplicacions instal·lades, la configuració, etc., i a les opcions avançades del CarismaTIC.

Per altra banda, ha de tenir un panell d'informació que mostri el nivell de bateria, l'estat de la connexió a internet, la data i l'hora. Per acabar, també ha de mostrar els logos de les dues entitats patrocinadores del CarismaTIC (La Fundació Ciutat de Viladecans i l'Obra Social de la Fundació "La Caixa") i d'i2CAT.

La *Vista Principal*, gestionada per la *MainActivity,* es pot veure a la Fig. 6, i a continuació es detallen els elements que hi destaquen.

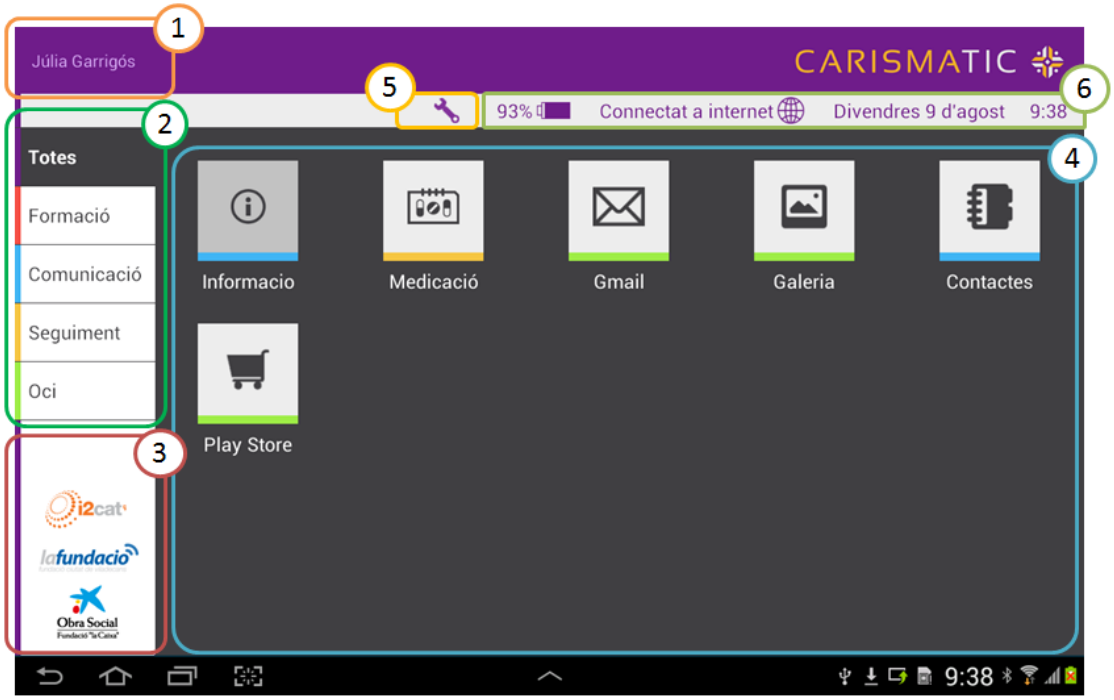

Fig. 6 Captura de la *Vista Principal* del Launcher remarcant els elements que la composen.

(1) **Nom de l'usuari** que ha iniciat sessió al dispositiu.
(2) **Menú lateral** que permet filtrar les aplicacions en funció de la seva categoria. A la Fig. 7 es pot veure la *Vista Principal* filtrant les aplicacions per la categoria *Comunicació.*

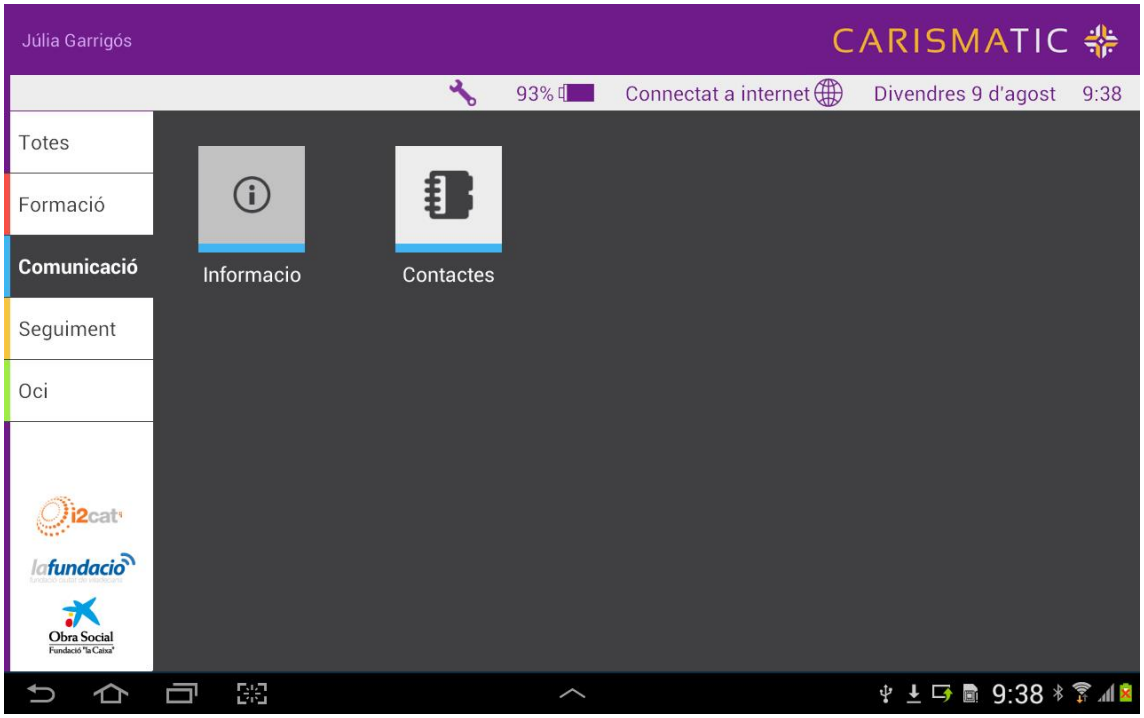

Fig. 7 *Vista Principal* del Launcher filtrant les aplicacions per la categoria *Comunicació.*

(3) **Logos** dels patrocinadors del projecte i d'i2CAT
(4) **Conjunt d'aplicacions del CarismaTIC** que han estat assignades a l'usuari. Cada una de les aplicacions està representada mitjançant el component de la Fig. 8, les parts del qual es descriuen tot seguit:

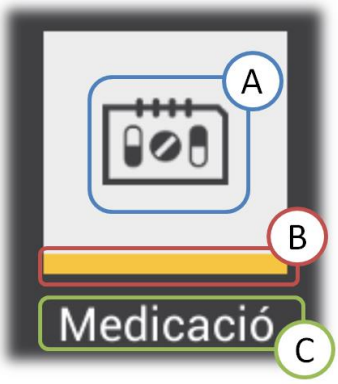

Fig. 8 Component que mostra la informació de cada aplicació i permet accedir-hi.

(A) Icona de l'aplicació.
(B) Barra de color que indica a quina categoria pertany l'aplicació.
(C) Títol de l'aplicació.

El fons d'aquest component varia en funció de l'estat de l'aplicació: si està instal·lada és blanc, tal com es veu a la Fig. 9 (b), si pel contrari no ho està, és gris com a la Fig. 9 (a). Quan una aplicació està instal·lada però té una actualització disponible es mostra una marca vermella al cantó superior dret que ho indica (veure Fig. 9(c)).

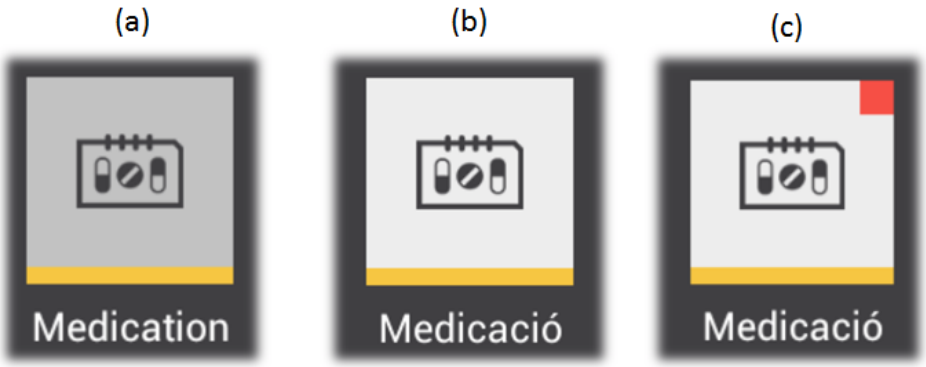

Fig. 9 Component que mostra la informació d'una aplicació. (a) Aplicació no instal·lada. (b) Aplicació instal·lada. (c) Aplicació instal·lada amb una actualització disponible.

(5) **Botó d'opcions**: obre un menú desplegable (veure Fig. 10) que permet accedir a *Totes les aplicacions* (és a dir, a la *Vista d'Aplicacions Instal·lades al Dispositiu*), a la *Configuració d'idioma* del dispositiu (el CarismaTIC farà servir l'idioma que estigui configurat al *tablet*), a la *Configuració del sistema* i finalment a la vista *d'Opcions avançades*. Aquest botó, a diferència de la resta, no està acompanyat de text que el descrigui, ja que porta a opcions que només han de ser accessibles per un usuari "expert" i, per tant, s'ha d'assegurar que el beneficiari no hi accedirà per error.

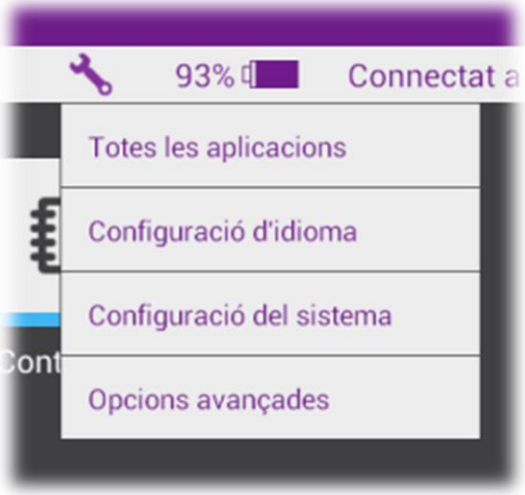

Fig. 10 Menú desplegable que apareix al polsar el botó d'opcions del Launcher.

(6) **Panell d'informació**: consta de tres components visuals (*Widgets*) que han estat dissenyats específicament per al CarismaTIC:

- *BatteryLevelIndicatorWidget*: consta d'un camp de text que mostra el nivell de bateria en percentatge i d'una imatge que el representa gràficament. A la Fig. 11 es poden veure les imatges que s'empren per mostrar els diferents nivells. Per mantenir el valor actualitzat, disposa d'un *BroadcastReceiver* que rep els canvis del nivell de bateria i modifica la imatge i el percentatge en conseqüència.

Fig. 11 Imatges que representen els diferents nivells de bateria al component *BatteryLevelIndicatorWidget*.

- **ConnectionIndicatorWidget**: consta d'un camp de text i d'una imatge que indiquen si hi ha o no connexió a internet. Per mantenir actualitzat l'estat, disposa d'un *BroadcastReceiver* que rep els canvis de connectivitat del dispositiu. Si el dispositiu està connectat a internet mostra el text i la imatge de la Fig. 12 (a). Pel contrari, si no ho està mostra el text i la imatge de la Fig. 12 (b).

Fig. 12 *Widget* que mostra l'estat de la connexió. (a) *Widget* quan hi ha connexió a internet. (b) *Widget* quan no hi ha connexió a internet.

- **CalendarAndClockWidget**: consta d'un camp de text que mostra la data i l'hora. Per mostrar la data actualitzada s'executa un *thread* que comprova periòdicament la data i l'hora i la modifica si és necessari. Aquest *thread* es finalitza quan es deixa de mostrar la *Vista Principal*.

3.2.3. Vista d'Aplicacions Instal·lades al Dispositiu

Els familiars o cuidadors del beneficiari, que probablement estaran més familiaritzats amb l'ús de *tablets,* han de poder accedir a qualsevol aplicació instal·lada al dispositiu. La *MenuActivity* s'encarrega de gestionar la vista que permetrà executar qualsevol aplicació instal·lada al dispositiu. La vista constarà, tal com es pot veure a la Fig. 13, d'una barra superior que permetrà *Sortir*, es a dir, tornar a la *Vista Principal* del Launcher, i d'un menú amb totes les aplicacions. Per a executar-ne una només cal polsar a sobre la seva icona.

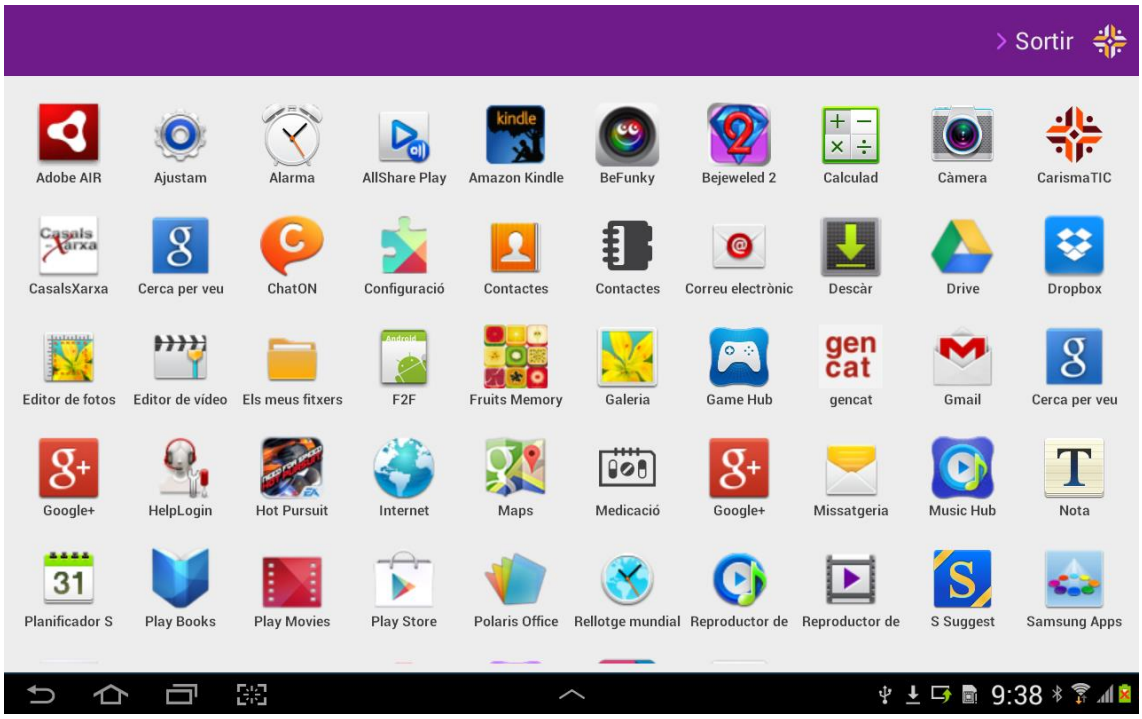

Fig. 13 Captura de la *Vista d'Aplicacions Instal·lades al Dispositiu*.

3.2.4. Finestra d'Opcions Avançades

Per assegurar que l'aplicació serà completament estable s'han considerat una sèrie d'errors, tot i que és molt poc probable que es produeixin. Si hi hagués algun problema en el registre de les notificacions *push* i el dispositiu no pogués rebre-les; l'usuari tindria l'opció de registrar-se i desregistrar-se de les notificacions *push* manualment. Per altra banda, si una notificació *push* indicant que cal actualitzar el llistat d'aplicacions no arribés al seu destí, llavors l'usuari podria actualitzar el llistat manualment.

A més, com que és possible que un dispositiu sigui emprat per un usuari i que al cap d'un temps sigui donat a un altre, és necessària una opció que permeti eliminar totes les dades confidencials d'un usuari perquè així l'altre no hi tingui accés. Quan es selecciona l'opció d'eliminar les dades confidencials del *tablet*, el Launcher torna a la *Vista d'Inici de Sessió*.

Tal com es pot veure a la Fig. 14, aquestes opcions es troben a la *Finestra d'Opcions Avançades*.

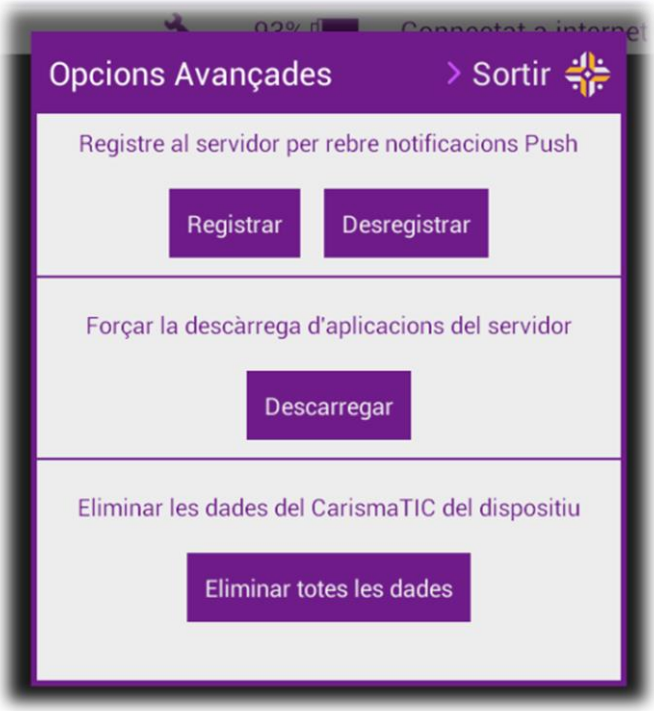

Fig. 14 Captura de la *Finestra d'Opcions Avançades*.

3.2.5. *Toasts* personalitzats

A Android, una de les maneres més habituals de mostrar missatges a l'usuari és mitjançant *toasts*, avisos que es mostren durant un temps i després desapareixen, se'n mostra un exemple a la Fig. 15 (a). Android permet triar entre dues duracions: 2 segons i 3,5 segons.

Aquestes duracions són massa curtes perquè una persona gran tingui temps de llegir els missatges. A més, Android no permet canviar el fons del *toasts*, la mida de la lletra, etc. És per aquest motiu que s'ha dissenyat un *toast* personalitzat per al CarismaTIC que es manté en pantalla 10 s i mostra els missatges de manera més clara (veure Fig. 15 (b)).

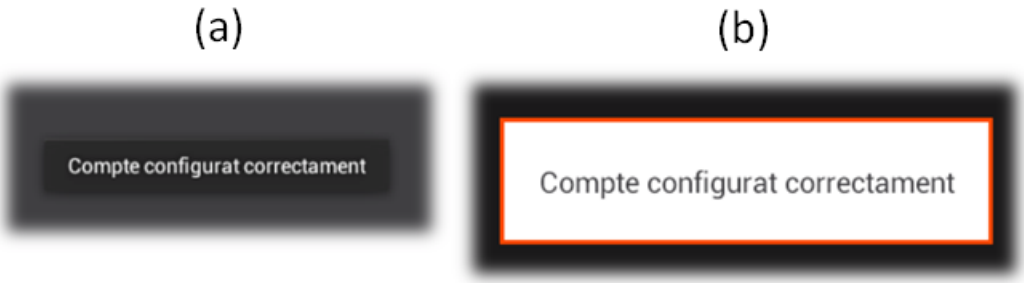

Fig. 15 Diferencia entre el toast d'Android (a) i el toast del CarismaTIC (b).

3.2.6. Finestra de *Loading*

Mentre l'aplicació està carregant dades, fent una petició, etc. és necessari mostrar un missatge que mostri a l'usuari que l'aplicació està treballant i no s'ha quedat penjada.

En alguns casos se sap l'estona que durarà el procés, com quan es tracta de la descàrrega d'un arxiu. Coneixent-ne la mida es pot mostrar una barra de progrés que vagi augmentant a mesura que es descarreguen les dades (veure la Fig. 16 (a)). El problema és que en molts casos no es pot establir la duració del progrés ni una manera de mostrar com avança. Llavors, es poden fer servir les barres indeterminades com la que es pot veure a la Fig. 16 (b).

Fig. 16 Finestra de *Loading*. (a) Finestra de *Loading* amb barra de progrés. (b) Finestra de *Loading* amb barra indeterminada.

3.2.6.1. *Barra de progrés de la descarrega de l'arxiu APK*

Una de les barres de progrés més importants del Launcher és la que mostra el procés de descàrrega de l'arxiu APK. Quan comença la descàrrega s'ha d'extreure la mida total de l'arxiu i anar augmentant la barra de progrés a mesura que es vagin descarregant els *bytes*.

El servidor, per motius d'eficiència, envia l'arxiu dividit en diferents parts; la qual cosa fa que no sigui possible esbrinar la mida total de l'arxiu i, per tant, que no es pugui mostrar la barra de progrés de manera correcta. Per solucionar-ho, es va decidir fer una primera petició al servidor demanant la mida de l'arxiu i a continuació descarregar l'arxiu. D'aquesta manera s'aconsegueix mostrar la barra de progrés, mantenint el sistema d'enviament en fragments del servidor.

3.3. Gestió d'aplicacions

El Launcher ha de permetre mostrar qualsevol tipus d'aplicació: aplicacions desenvolupades per al CarismaTIC, aplicacions de tercers (que s'extrauran del Play Store) i aplicacions que incorporen els dispositius de sèrie com el Calendari, la Calculadora, la Càmera, etc.

3.3.1. Instal·lació d'aplicacions

Un dels requeriments del projecte és que el Launcher instal·li al dispositiu les aplicacions del llistat que encara no ho estiguin. Tanmateix, Android no permet instal·lar aplicacions sense interacció de l'usuari per motius de seguretat. Per tant, el Launcher haurà de facilitar al màxim la instal·lació de les aplicacions. Hi ha dues maneres de fer-ho. La primera opció consisteix en obrir el Play Store amb la pàgina que conté informació de l'aplicació, on el beneficiari podrà veure captures de l'aplicació i instal·lar-la. Per procedir a la instal·lació, caldrà que premi el botó *Instal·lar* (veure Fig. 17) i que després confirmi que vol instal·lar l'aplicació acceptant els permisos que aquesta requereix, tal com es veu a la Fig. 18.

Fig. 17 Captura de l'aplicació Play Store en què es pot veure la pàgina d'informació de l'aplicació *Frutas Juego de Memoria*. S'ha remarcat en vermell el botó que permetrà a l'usuari instal·lar l'aplicació.

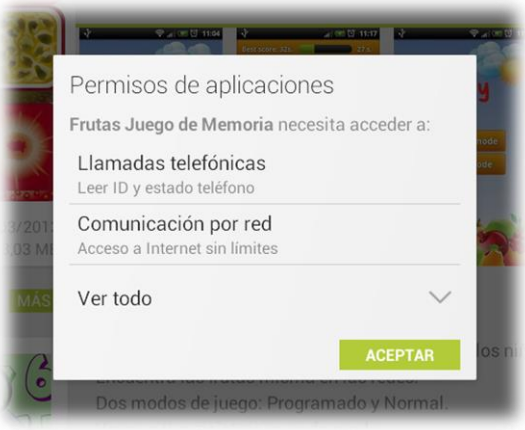

Fig. 18 Finestra que es mostra quan l'usuari prem el botó *Instal·lar*. Llista els permisos que empra l'aplicació, i un cop són acceptats aquesta és instal·lada.

La segona opció consisteix en descarregar l'arxiu d'instal·lació (APK) i executar-lo de manera que directament aparegui la finestra de confirmació de la instal·lació (veure Fig. 19). Un cop s'ha instal·lat l'aplicació es mostra una finestra que permet tornar on es trobava l'usuari o obrir l'aplicació que acaba de ser instal·lada.

Aquesta última és la manera més intuïtiva d'instal·lar una aplicació, ja que requereix menys passos i la interfície és més senzilla. Tanmateix, el Launcher només pot instal·lar aplicacions d'aquesta manera si té accés als seus arxius APK. Com que el Play Store no ofereix cap servei per descarregar aquests arxius, aquesta opció no es podrà fer servir per instal·lar les aplicacions desenvolupades per tercers.

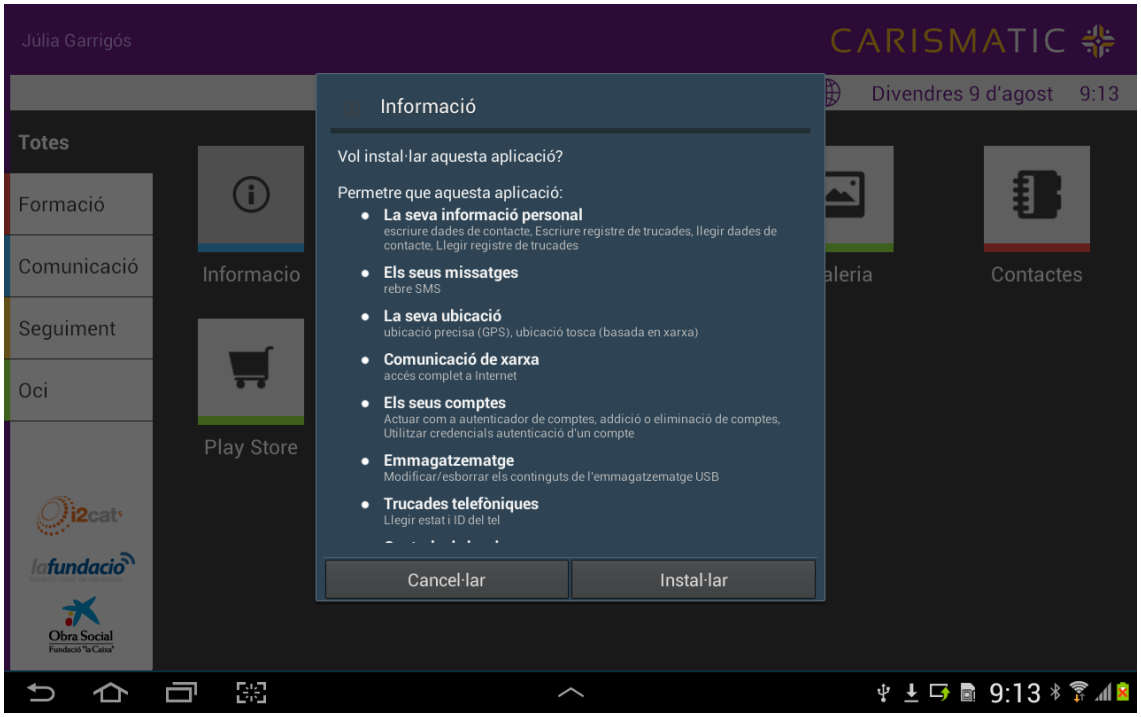

Fig. 19 Finestra de confirmació de la instal·lació d'una aplicació. Mostra a l'usuari els permisos que aquesta demana i li permet cancel·lar o realitzar la instal·lació.

Les aplicacions desenvolupades per al CarismaTIC podran instal·lar-se descarregant l'APK i les programades per tercers s'hauran d'instal·lar a través del Play Store. Per fer-ho, el Launcher obre el Play Store per la pàgina de l'aplicació tal com es veu a la Fig. 17 i permet així instal·lar-la.

3.3.2. Execució d'aplicacions

Per a executar una aplicació que està instal·lada al dispositiu només cal el seu *PackageName*, que és el camp que la identifica de manera unívoca al Play Store.

3.3.3. Actualització d'aplicacions

Pel que fa a l'actualització d'aplicacions, si aquestes es troben al Play Store, independentment d'on hagin estat descarregades, s'actualitzaran de forma automàtica. Tanmateix, en general al CarismaTIC les aplicacions no seran penjades al Play Store i, per tant, s'haurà de trobar una alternativa per actualitzar-les.

La solució que s'ha trobat consisteix en assignar a totes les aplicacions desenvolupades per al CarismaTIC un camp *Version*, que indicarà la versió més nova de l'aplicació. Així, el Launcher només caldrà que comprovi si la versió instal·lada coincideix amb la més nova. Si no, donarà l'opció d'actualitzar-la. Per fer-ho, es descarregarà l'APK més recent i s'instal·larà igual que quan es tracta d'una aplicació nova.

3.3.4. Tipus d'aplicacions

Com s'ha vist, el Launcher haurà de suportar diferents tipus d'aplicacions que seran tractades de manera diferent. Les característiques de les aplicacions es definiran als objectes *LauncherApp.* Aquests objectes contindran la informació que necessita el Launcher per mostrar l'aplicació, permetre instal·lar-la, etc.

L'administrador del sistema, que és l'encarregat de registrar les aplicacions a la plataforma, farà servir un *backoffice* per afegir les aplicacions que després els metges assignaran als beneficiaris. Les dades que haurà d'introduir variaran en funció del tipus d'aplicació que estigui registrant.

A continuació es mostren les dades que són comunes a tots els tipus d'aplicacions. Després es detallaran els diferents tipus d'aplicacions i els camps concrets que necessiten.

- ***Title***: és el nom de l'aplicació, que es fa servir per identificar-la. Al Launcher, mentre l'aplicació no està instal·lada, mostra aquest text com a títol. Un cop s'instal·la ja no es fa servir, ja que s'extreu el títol de l'aplicació. D'aquesta manera, si el títol varia amb l'idioma, com passa amb l'aplicació Contactes que en castellà es diu *Contactos*, es mostra en la llengua corresponent.
- ***Icon***: icona de l'aplicació que mostrarà el Launcher.
- ***Category***: indica el servei del CarismaTIC al que correspon l'aplicació, serveix per mostrar les aplicacions ordenades al Launcher. Pot prendre quatre valors: formació, comunicació, seguiment i oci.
- ***Description***: es farà servir al web per descriure l'aplicació.

Als apartats següents s'expliquen les característiques concretes de cada tipus d'aplicació que suporta el Launcher.

3.3.4.1. *Aplicacions del Play Store*

La plataforma ha de suportar aplicacions desenvolupades per tercers. Per a registrar-les, només cal introduir el seu *PackageName*, que és el camp que l'identifica de forma unívoca al Play Store.

Tal com s'ha vist, aquest tipus d'aplicacions s'instal·laran i actualitzaran a través del Play Store, de manera que no caldrà que el Launcher en faci un control de versions.

3.3.4.2. *Aplicacions d'i2CAT*

Les aplicacions desenvolupades expressament per al CarismaTIC s'instal·laran i actualitzaran descarregant els seus corresponents arxius APK del servidor.

Android proporciona un sistema que permet que dues aplicacions independents es comuniquin. Això és molt útil, per exemple, per comunicar l'aplicació de contactes i la de videoconferència, que són aplicacions independents perquè s'han desenvolupat emprant tecnologies diferents. Per a executar una aplicació només fa falta conèixer la seva URI, que es forma amb l'*scheme,* que s'haurà definit a l'*AndroidManifest.xml* de l'aplicació, i els paràmetres i valors d'aquest (veure Fig. 20).

```
scheme://param1=value1&param2=value2
```

Fig. 20 Esquema de la URI necessària per executar una aplicació amb paràmetres.

Les aplicacions d'i2CAT es divideixen en dos tipus: aplicacions que han de ser executades mitjançant la URI per poder funcionar (aplicacions que sense paràmetres no funcionen) i aplicacions que no necessiten paràmetres.

Per a introduir a la plataforma una aplicació sense paràmetres, a més de les dades comunes, cal l'APK de l'aplicació. El sistema s'encarregarà d'extreure el *PackageName* i la *Version* de dins.

Pel que fa a les aplicacions que necessiten paràmetres, com que es tracta d'aplicacions que només es poden executar si se'ls passen paràmetres, s'ha d'evitar que l'usuari les executi. Per això, un cop s'instal·len es deixen de mostrar al Launcher. Per introduir a la plataforma aquest tipus d'aplicacions, a més de les dades comunes cal:

- L'APK de l'aplicació del qual s'extraurà el *PackageName* i la *Version*
- Nom dels paràmetres que s'hauran de passar a l'aplicació perquè sigui executada.
- *Title*: en aquest cas el *title* té més importància perquè definirà l'*scheme* de l'aplicació (veure Fig. 20).

3.3.4.3. *Aplicacions de sèrie (*default*)*

Tots els dispositius Android incorporen aplicacions de sèrie (que varien en funció del model): la calculadora, el calendari, el correu electrònic, els mapes, etc. Aquestes aplicacions no tenen el mateix *PackageName* a tots els models de dispositius i no es poden executar de manera normal.

Per a executar una aplicació *default* fan falta dos camps que les identifiquen a tots els dispositius Android: el *SelectorAction* i el *SelectorCategory.* Per exemple, per obrir l'aplicació de Calendari farien falta les dades següents:

- *SelectorAction*: android.intent.action.MAIN
- *SelectorCategory*: android.intent.category.APP_CALENDAR

El *SelectorAction* pren sempre el mateix valor mentre que el *SelectorCategory* identifica l'aplicació.

3.3.4.4. *Càmera*

Pel que fa a l'aplicació de la càmera, no hi ha un *SelectorCategory* que permeti executar-la. Per això s'ha hagut de crear una funció al Launcher que, per a cada dispositiu, identifica el *PackageName* de la càmera.

3.4. Arquitectura del Launcher

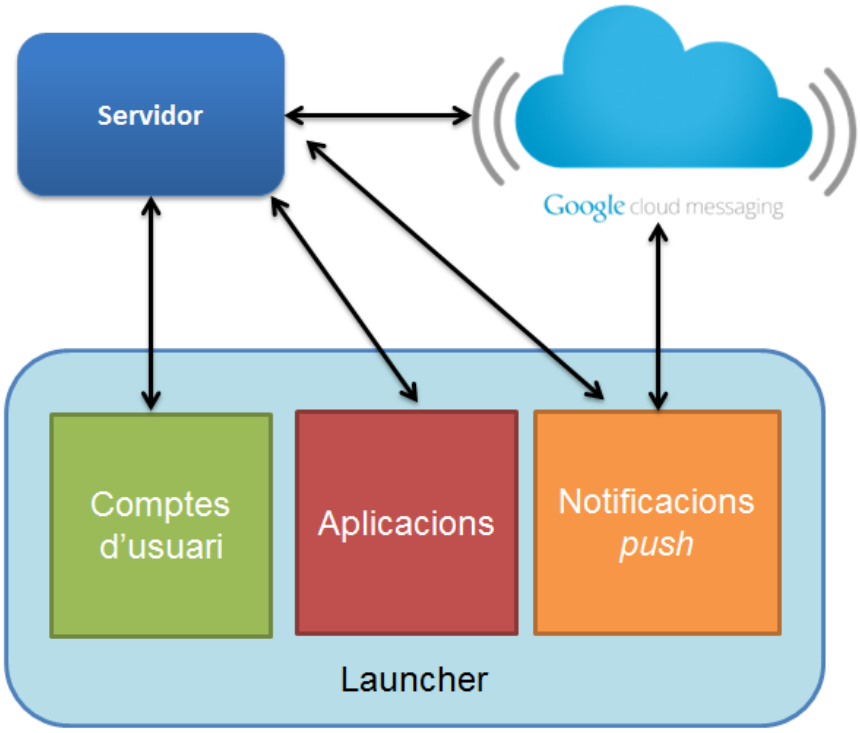

Fig. 21 Arquitectura general del Launcher.

El Launcher consta de tres grans mòduls (veure Fig. 21). El mòdul de comptes d'usuari, que configurarà un compte al dispositiu i donarà accés a la resta d'aplicacions per aconseguir l'*authtoken* que els permetrà autenticar-se al servidor. El mòdul d'aplicacions que obtindrà el llistat d'aplicacions del servidor, el guardarà al dispositiu, comprovarà quines aplicacions hi ha instal·lades i mostrarà el llistat en pantalla. I per últim, el mòdul que gestionarà les notificacions *push*, és a dir: el registre, el des-registre, la recepció de les notificacions, etc.

Cada un d'aquests mòduls es comunica amb el servidor per obtenir informació. El mòdul de notificacions, a més, es comunica amb el servei *Google Cloud Messaging* per a realitzar el registre, tal com s'explica a l'apartat B.1.5.

Hi ha tres classes comunes que poden fer servir tots els mòduls:

- ***NetworkUtilities***: inclou totes les funcions que accedeixen a internet, que han de ser executades sempre de manera asíncrona, és a dir, des d'un *thread* diferent al principal.
- ***FileUtilities***: inclou les funcions que permeten guardar arxius a la memòria del dispositiu i extreure'ls quan fa falta emprar-los.
- ***LauncherUtilities***: inclou totes les funcions que necessitaran les aplicacions que s'integrin al Launcher per funcionar. Permet extreure el compte d'usuari, comprovar si una aplicació està instal·lada, etc.

Als següents apartats s'explicarà en detall el funcionament de cada un dels mòduls principals.

3.4.1. Mòdul 1: Comptes d'usuari

Android incorpora un sistema per a gestionar comptes d'usuari que guarda les credencials i permet a altres aplicacions accedir-hi. L'objectiu d'aquest sistema és permetre als dispositius Android accedir a serveis OAuth2.

3.4.1.1. *OAuth2 a Android*

OAuth2 permet a una aplicació accedir a un servei d'un tercer. Un exemple seria una aplicació que vol accedir a *Facebook* per publicar al mur de l'usuari. Primer, l'usuari haurà d'autoritzar l'aplicació per a accedir en nom seu a *Facebook* (es suposa que l'usuari té configurat al dispositiu el compte de *Facebook*). Un cop fet això l'aplicació pot demanar un *authtoken* que li permeti accedir en nom de l'usuari a *Facebook*.

Un *authtoken* és una cadena de caràcters que s'obté mitjançant el nom d'usuari i la contrasenya de l'usuari. Aquest permet accedir al servei en qüestió en nom de l'usuari durant un temps limitat. Un cop ha passat aquest temps l'authtoken queda invalidat i se n'ha de demanar un de nou.

L'AccountManager d'Android s'encarrega de guardar les credencials de l'usuari i de demanar l'*authtoken* quan una aplicació ho sol·licita. Mitjançant aquest sistema es poden fer servir les credencials d'un servei en una aplicació d'un tercer sense que aquesta aplicació arribi a tractar mai amb les credencials de

l'usuari, garantint-ne així la seguretat i confidencialitat. L'única dada que necessita l'aplicació per accedir al servidor és l'*authtoken*, que és aconseguit per l'AccountManager amb les credencials que té emmagatzemades.

3.4.1.2. *Autenticació i obtenció d'authtokens*

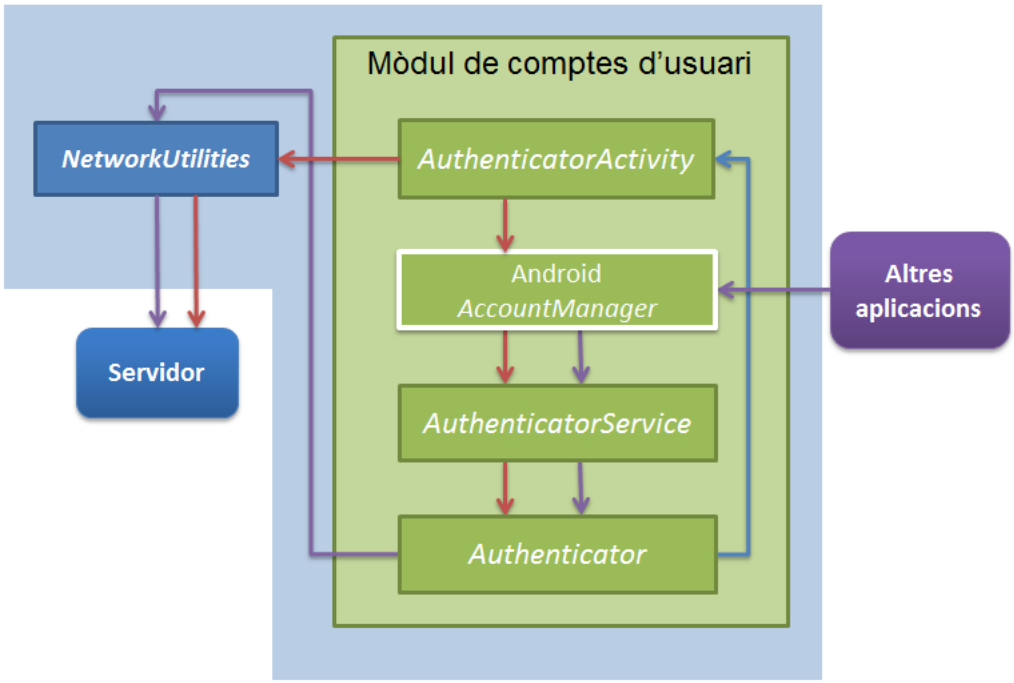

Fig. 22 Estructura del mòdul de comptes d'usuari i comunicacions que es produeixen.

L'AccountManager d'Android necessita una classe, anomenada *Authenticator,* que s'encarregui de les gestions del compte del CarismaTIC. Amb l'objectiu que qualsevol aplicació pugui fer gestions amb aquest compte, el servei *AuthenticationService*, que s'encarrega de retornar l'*Authenticator*, estarà sempre actiu. L'*Authenticator* haurà de fer peticions d'*authtoken* al servidor, permetre confirmar les credencials, canviar-les, etc.

El Launcher ha de permetre iniciar sessió introduint el nom d'usuari i la contrasenya el primer cop que s'accedeix a la plataforma. També ha de proporcionar a altres aplicacions l'*authtoken* que els permetrà emprar els serveis del CarismaTIC.

A la Fig. 22 es mostra en vermell el primer cas, en el qual l'*AuthenticatorActivity* mostra un formulari on l'usuari introdueix les seves credencials (veure Fig. 5). Un cop s'han introduït, fa una petició de *Login* al servidor emprant el *NetworkUtilities*. Si el servidor accepta les credencials com correctes, accedeix a l'*AccountManager* (que, per la seva part, accedeix a l'*AuthenticatorService,* i aquest a *l'Authenticator*) i introdueix les dades del compte: tipus de compte, nom d'usuari, contrasenya i *authtoken*. Amb això, al

dispositiu s'ha configurat el compte, que serà accessible per totes les aplicacions que tinguin els permisos corresponents.

Per aconseguir l'*authtoken*, les aplicacions hauran d'accedir a l'*AccountManager* (fletxes en lila de la Fig. 22), que, si ja té un *authtoken*, el retornarà directament. Si aquest fos invàlid, s'hauria de tornar a fer la petició a l'*AccountManager* indicant que s'ha d'invalidar l'*authtoken* anterior. Quan s'invalida un *authtoken*, l'*AccountManager* accedeix a l'*AuthenticationService*, que retorna una instància de l'*Authenticator*; que fa llavors la petició d'*authtoken* al servidor (a través del *NetworkUtilities*).

Si al demanar l'*authtoken* al servidor les credencials no fossin acceptades, l'*Authenticator* mostraria un altre cop el formulari d'inici de sessió (mitjançant l'*AuthenticatorActivity*) perquè l'usuari pogués tornar a introduir les credencials (fletxes en blau de la Fig. 22).

3.4.2. Mòdul 2: Aplicacions

Com ja s'ha vist, hi ha dues vistes al mòdul d'aplicacions: la *Vista Principal*, que mostrarà les aplicacions del CarismaTIC, i la *Vista d'Aplicacions Instal·lades al Dispositiu*.

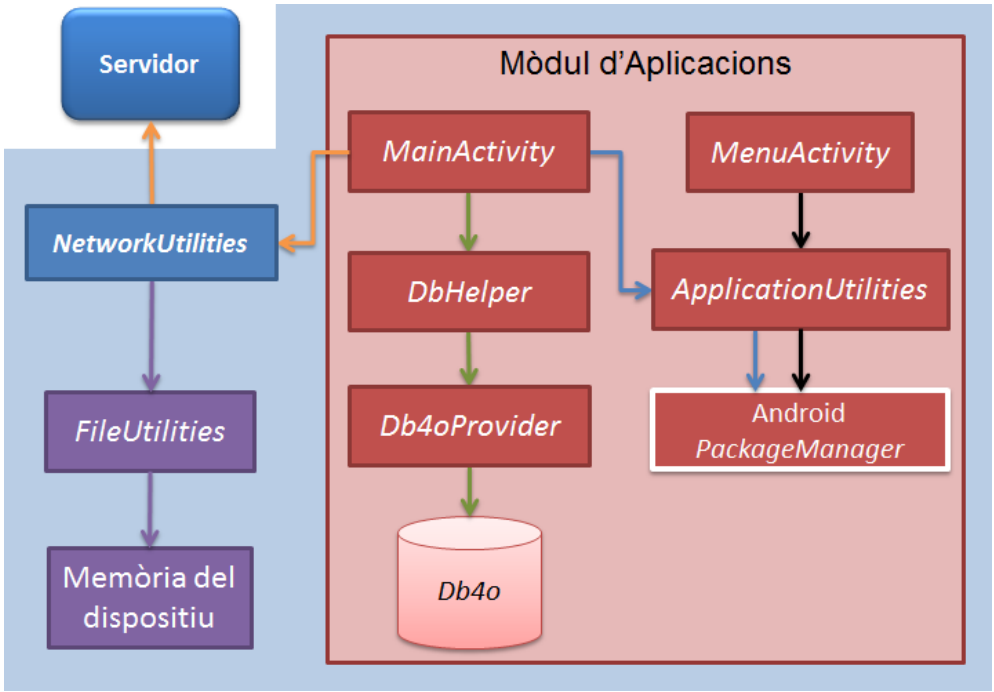

Fig. 23 Estructura del mòdul d'aplicacions i interacció entre els components.

La *MainActivity* mostrarà la vista principal del Launcher, que conté les aplicacions triades per al beneficiari. Primer haurà d'aconseguir el llistat d'aplicacions fent la petició al servidor a través del *NetworkUtilities* (veure fletxes taronges). Quan el *NetworkUtilities* rebi el llistat, descarregarà la icona

de cada aplicació i la passarà al *FileUtilities* perquè la guardi a la memòria del dispositiu (veure fletxes liles).

Un cop s'obtingui el llistat d'aplicacions s'haurà de guardar a la base de dades. Per fer-ho, la *MainActivity* accedirà al *DbHelper*, i aquest al *Db4oProvider*, que és qui realitzarà les interaccions amb la base de dades Db4o (veure fletxes verdes). El motiu pel qual la *MainActivity* no es comunica directament amb el *Db4oProvider* és abstraure-la del tipus de base de dades que s'està emprant. D'aquesta manera, si es volgués canviar la base de dades per una del tipus SQLite, es podria fer canviant només el *DbHelper* i no caldria modificar la *MainActivity.*

Per a mostrar una aplicació, la *MainActivity* haurà de tenir en compte si aquesta està instal·lada al dispositiu, si està actualitzada, de quin tipus d'aplicació es tracta, etc. Per accedir a aquesta informació, fa servir l'*ApplicationUtilities* (veure fletxes blaves), que dóna totes les funcions que necessita el Launcher per mostrar les aplicacions. Per a poder fer-ho, accedeix al *PackageManager* d'Android, que proporciona totes les dades de les aplicacions instal·lades al dispositiu.

Per altra banda, la *MenuActivity serà l'encarregada de* mostrar les aplicacions instal·lades al dispositiu. Per a fer-ho, demana a l'*ApplicationUtilities* el llistat d'aplicacions. Aquest accedeix al *PackageManager* i retorna el llistat d'aplicacions amb el seu títol, *PackageName*, icona, etc. (veure fletxes negres). D'aquesta manera, la *MenuActivity* pot mostrar el menú amb totes les aplicacions (veure Fig. 13).

3.4.3. Mòdul 3: Notificacions *Push*

Per a poder rebre notificacions *push,* cal que el Launcher es registri al servidor de *Google Cloud Messaging* (GCM) i que doni la informació d'aquest registre al servidor del CarismaTIC (tal com s'explica a l'apartat B.1.5). D'aquesta manera el servidor li podrà enviar notificacions a través de GCM.

La *MainActivity*, quan s'inicia, demana al *GCMRegistrar* que registri el dispositiu al servidor GCM si encara no ho està (veure fletxes en verd de la Fig. 24). La resposta del GCM es rep al *GCMReceiver,* que envia la resposta al *GCMIntentService* perquè realitzi les accions pertinents. Quan el GCM respon que el registre ha estat correcte (veure fletxes en vermell), el *GCMIntentService* indica al *PushUtilities* que s'ha de registrar l'ID rebut del GCM al servidor del *CarismaTIC*. El *PushUtilities*, a través del *NetworkUtilities*, realitza aquest registre.

Quan el servidor vol enviar a un dispositiu una notificació *push* (veure fletxes en blau), envia el missatge al GCM indicant l'ID del dispositiu destí. Llavors, el GCM envia la notificació, que és rebuda pel *GCMReceiver*, que la transmet al *PushUtilities* a través del *GCMIntentService*. El *PushUtilities* s'encarrega de la

gestió de la notificació, que dependrà de seva prioritat, tal i com s'explica al següent subapartat.

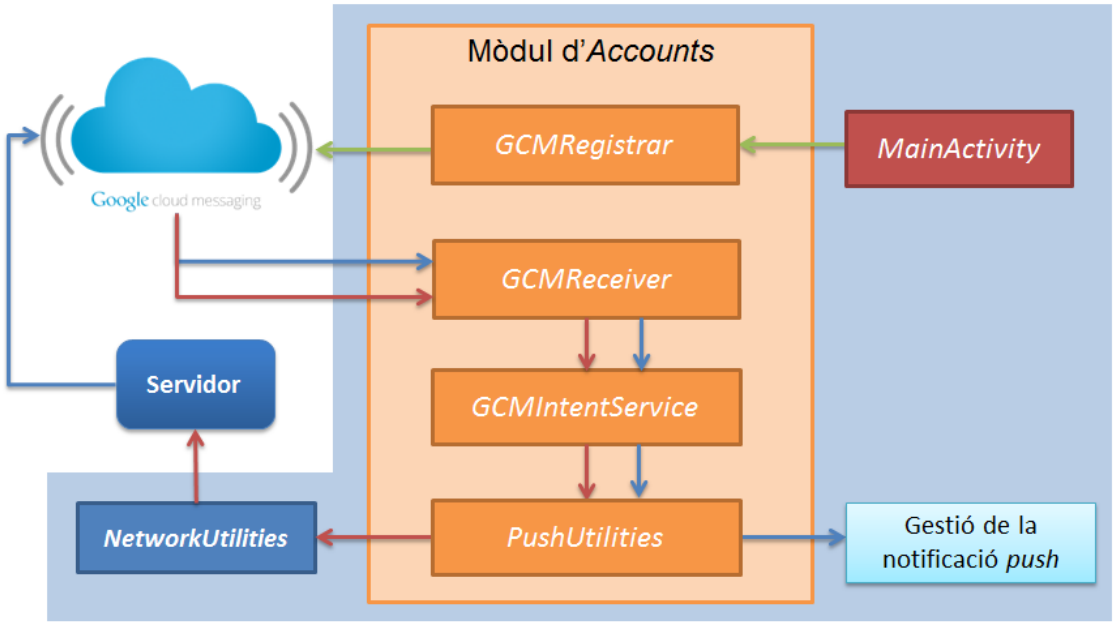

Fig. 24 Estructura del mòdul encarregat de la gestió de les notificacions *push*.

3.4.3.1. *Prioritització de les notificacions* push

El Launcher rebrà diferents tipus de notificacions *push*: trucades entrants, missatges de text, notificacions que indicaran que cal actualitzar el llistat d'aplicacions, etc. Cada una d'aquestes notificacions ha de ser tractada de manera diferent, per la qual cosa s'han definit tres nivells de prioritat:

- **Prioritat alta**: les úniques notificacions que s'han definit fins ara com a molt prioritàries són les de trucades entrants. És important que, quan es rep aquesta notificació, s'avisi ràpidament a l'usuari perquè pugui respondre-hi. Quan es rep una notificació amb aquesta prioritat, el *PushUtilities* executa la *CallingMeActivity*, que mostra a l'usuari una finestra indicant qui està trucant mentre reprodueix el so d'un telèfon. Permet a l'usuari acceptar o rebutjar la trucada, tal com es pot veure a la Fig. 25. A l'apartat 4.3 es detalla el funcionament de les trucades.

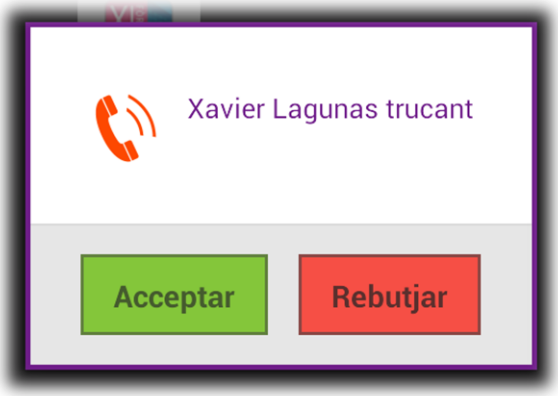

Fig. 25 Finestra que mostra l'avís de trucada entrant. Aquesta vista és controlada per la *CallingMeActivity.*

- **Prioritat mitja**: són notificacions que no cal que l'usuari rebi immediatament però que són importants per al funcionament del Launcher. Per exemple, les que indiquen que cal actualitzar el llistat d'aplicacions. El *PushUtilities* enviarà un missatge en *broadcast* que, si és rebut per la *MainActivity,* actualitzarà les aplicacions. Però si en aquell moment l'usuari no es trobés al Launcher o tingués el dispositiu bloquejat, la *MainActivity* no rebria el *broadcast*. Per assegurar que quan es torni a executar la *MainActivity,* s'actualitzarà el llistat d'aplicacions, el *PushUtilities* guardarà un paràmetre a la configuració indicant que s'ha rebut aquesta notificació de manera que quan la *MainActivity* torni a estar activa veurà que cal actualitzar.

- **Prioritat baixa**: són notificacions que, si no es reben en el moment que arriben, no té sentit que es rebin. Un exemple d'aquest tipus de notificació seria la cancel·lació d'una trucada: si encara s'està mostrant l'avís de trucada entrant a l'usuari, té sentit mostrar el missatge indicant que la trucada ha estat cancel·lada. En canvi, si no s'està mostrant l'avís, no té sentit mostrar el missatge. Per això, el *PushUtilities* enviarà un missatge en *broadcast* indicant que s'ha rebut la notificació però no ho guardarà a cap paràmetre de configuració. Així, si *l'Activity* que l'ha de rebre està inactiva, no rebrà el missatge i aquest es perdrà.

Fer servir prioritats en les notificacions permetrà que, més endavant, quan hi hagi molts tipus de notificacions es puguin gestionar de manera fàcil. Si no hi hagués prioritats s'hauria de tractar cada tipus de notificació individualment, la qual cosa no seria eficient.

3.4.3.2. *Alertes*

Als *tablets* Android, quan es rep una notificació es reprodueix un so i es mostra una alerta a la part inferior dreta de la pantalla, tal com es pot veure a la Fig. 26, en què una alerta informa que s'acaba de rebre un correu electrònic.

Aquest avís, al cap de pocs segons, desapareix de la pantalla, i per accedir a la informació que conté s'ha d'obrir el centre de notificacions (veure Fig. 27).

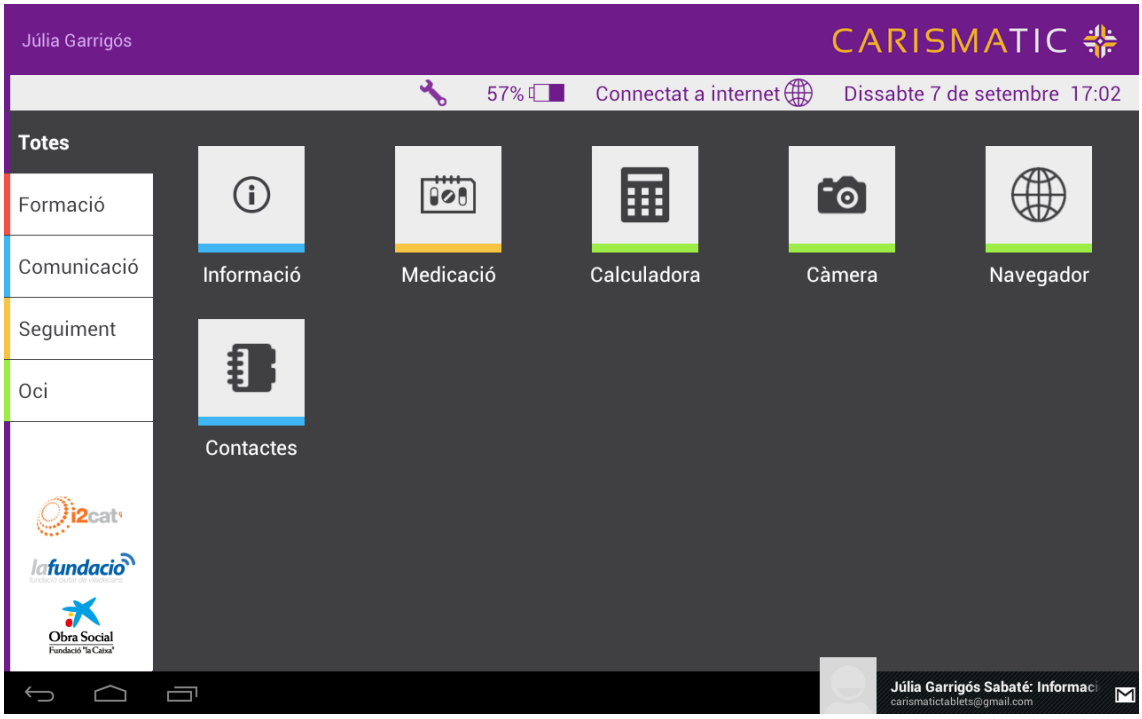

Fig. 26 Captura de la *Vista Principal* just quan s'acaba de rebre un correu electrònic.

Fig. 27 Captura del centre de notificacions que mostra la informació de la notificació rebuda.

Aquestes alertes no estan pensades perquè les rebi una persona gran. Tampoc estan preparades per a alertar d'una trucada entrant, ja que no permeten triar el so que es reprodueix ni poden aparèixer al mig de la pantalla per cridar l'atenció de l'usuari. Per aquests motius s'ha creat un sistema d'alerta específic pel CarismaTIC. Ara mateix, només es fa servir per alertar de trucades

entrants, quan es rep la notificació es comença a reproduir el so d'un telèfon mentre es mostra l'avís de la Fig. 25. El so segueix sonant fins que l'usuari tria una opció o la trucada caduca (a l'apartat 4.3 es veurà quan passa això).Si el dispositiu, en el moment de rebre la notificació, es troba en una altra aplicació, es força que es mostri l'avís en pantalla. Si el dispositiu està bloquejat quan es rep la notificació, es desbloqueja i es mostra l'avís.

3.5. Funcionament del Launcher

Al diagrama de flux de la Fig. 28 es mostra el procés que segueix el Launcher quan s'inicia.

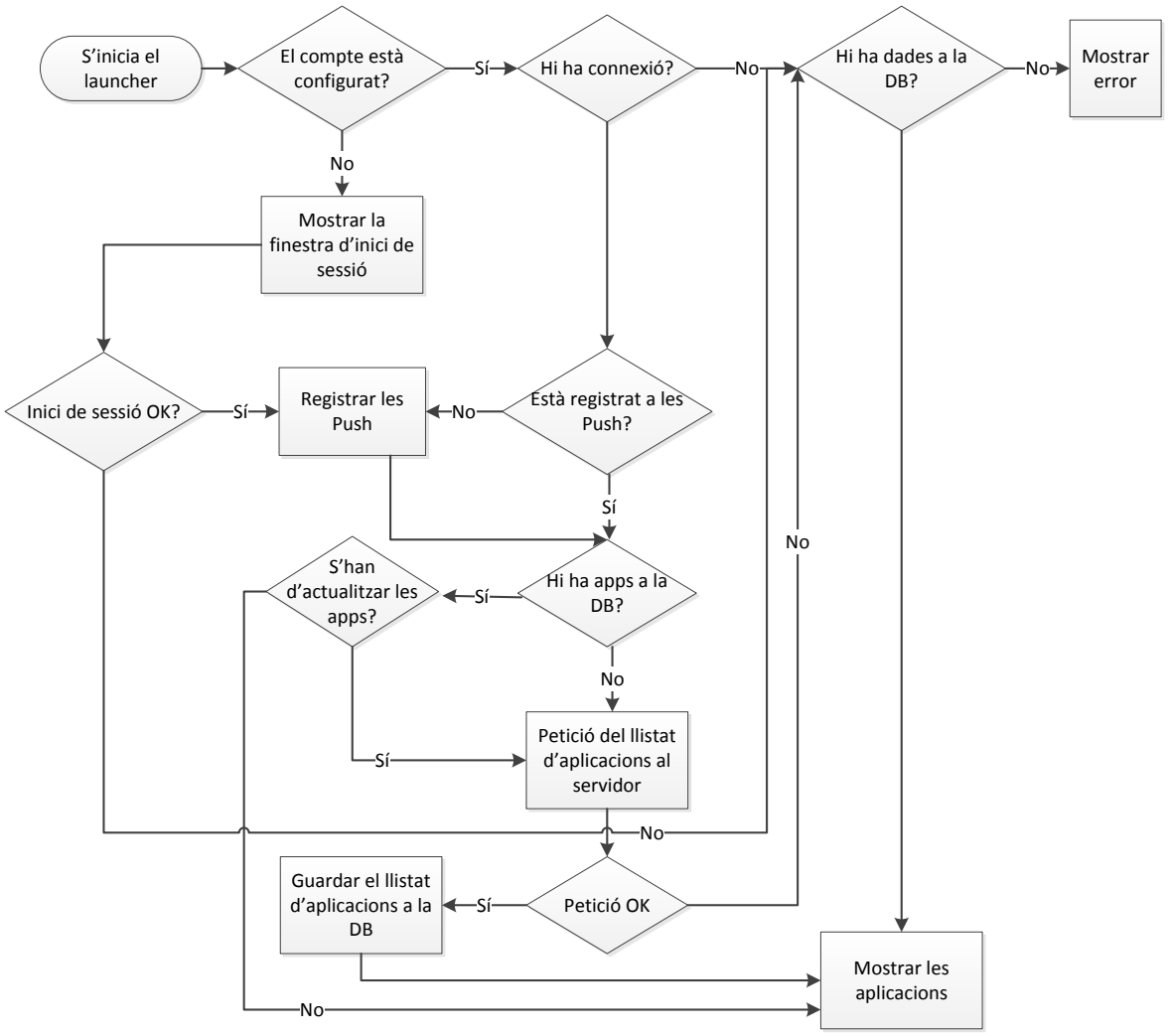

Fig. 28 Diagrama de flux de l'inici del Launcher.

Per començar comprova si el compte del CarismaTIC està configurat i, si no ho està, mostra la vista d'*Inici de sessió*. Si el compte està correctament configurat, passa a comprovar l'estat de la connexió a internet. Si la connexió

és correcta comprova el registre de les *push* (si el dispositiu no està registrat, el registra) i comprova si hi ha aplicacions a la base de dades. Si n'hi ha, comprova si s'ha rebut una notificació *push* indicant que cal actualitzar-les. Si és així fa la petició al servidor, guarda les aplicacions a la base de dades i les mostra (mitjançant la *Vista Principal)*. En cas contrari mostra les aplicacions de la base de dades en pantalla.

Si el dispositiu no té connexió, comprova si hi ha dades a la base de dades, si és així, les mostra, i en cas contrari mostra un error a l'usuari.

CAPÍTOL 4. DISSENY I IMPLEMENTACIÓ DEL SISTEMA DE COMUNICACIÓ

El Launcher es comunicarà amb el servidor del CarismaTIC a través d'HTTPS per assegurar que les comunicacions són segures, i s'autenticaran totes les comunicacions mitjançant *authtokens* tal com s'ha vist a l'apartat 3.4.1. Els *Controllers* de *Spring* (veure apartat B.5) s'encarregaran de rebre les peticions dels *tablets*.

En aquest capítol s'explicarà com s'ha dut a terme la implementació de comunicacions HTTPS, el funcionament dels *Controllers* que es comuniquen amb els dispositius mòbils i els protocols que es fan servir.

4.1. Implementació de comunicacions HTTPS a Android

En el marc d'aquest projecte, ha sorgit un problema en voler establir connexions HTTPS amb el servidor. Com que el certificat del CarismaTIC és provisional, no ha estat emès per una entitat de confiança, per la qual cosa Android no hi confia i no permet establir-hi connexions.

Per aconseguir que Android confiï en el servidor mentre el projecte es troba en fase de desenvolupament, s'ha hagut d'instal·lar el certificat del servidor al dispositiu. Android només suporta certificats *Bouncy Castle* (veure [1]), que tenen l'extensió BKS, per la qual cosa ha calgut convertir el certificat del servidor, que estava en format CRT, a aquest format.

El *NetworkUtilities* s'encarrega de gestionar totes les connexions sortints del *tablet*. Ho fa mitjançant dues classes diferents: *HttpClient* per a les peticions que retornen una cadena de caràcters i *HttpURLConnection* per a les peticions que retornen arxius. Per una banda, s'ha hagut de crear la classe *CarismaticHTTPSClient*, que realitza la mateixa funció que l'*HttpClient* però confiant en el certificat del CarismaTIC. Per altra banda, s'ha passat a fer servir la classe *HttpsURLConnection* per a les descàrregues d'arxius i s'ha hagut de configurar perquè confiï en el certificat del servidor.

4.2. Gestió de les peticions del *tablet*

El *MobileController* és l'encarregat de gestionar totes les peticions provinents del *tablet* excepte les relacionades amb trucades, la gestió de les quals es descriu a l'apartat 4.3.

El *MobileController* fa ús de tres *Services* que li permeten contestar les peticions del *tablet*; a la Fig. 29 es poden observar les comunicacions que es produeixen. Als apartats següents s'expliquen les funcions que aporten el *PushService* i *l'AuthService*. Per la seva banda, el *UserService,* accedeix a la base de dades a través del *UserRepository* i n'extreu les dades d'usuari

necessàries. No s'aprofundirà més en el seu funcionament perquè no forma part d'aquest projecte.

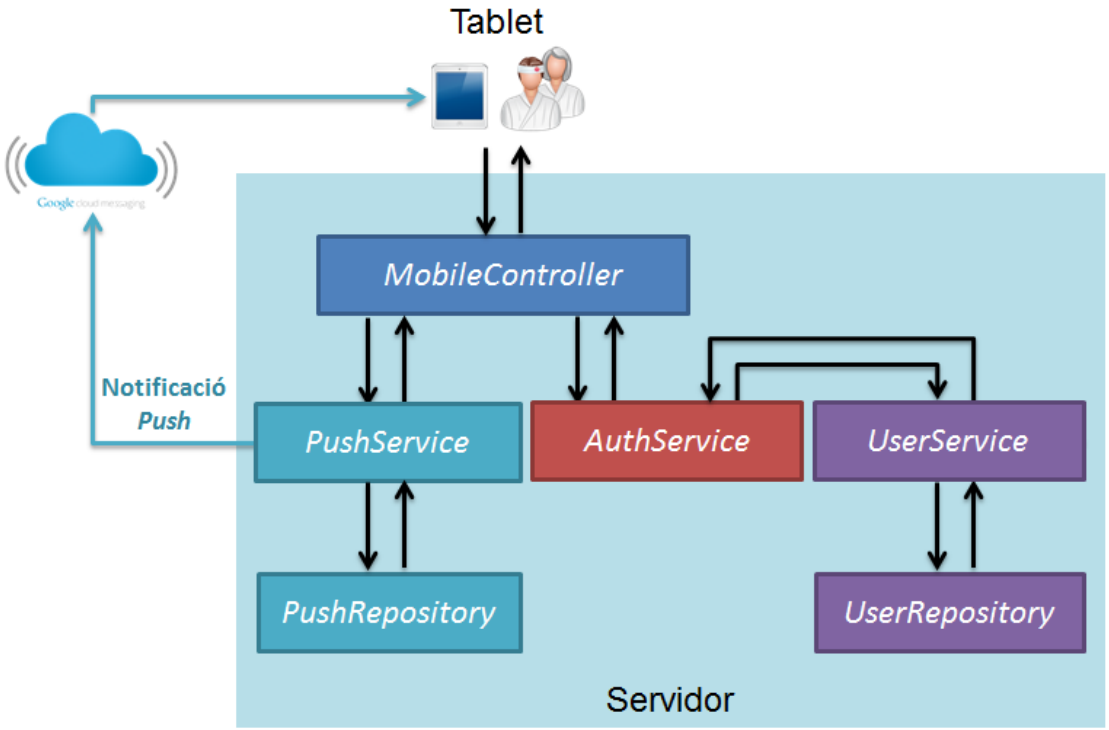

Fig. 29 Esquema de comunicacions del *tablet* amb el servidor. Es pot veure que el *MobileController* es comunica amb diferents *Services* i que alguns d'aquests accedeixen a *Repositories* (que són els components encarregats d'accedir a la base de dades).

4.2.1. Servei d'autenticació (*AuthService*)

Com ja s'ha vist, és necessari autenticar totes les peticions que es facin al servidor, i per aconseguir-ho s'empraran *Authtokens* (veure apartat 3.4.1.1). Els *authtokens* es generaran a l'*AuthService*, que també els haurà de validar cada cop que es faci una petició.

4.2.1.1. *Obtenció de l'authtoken*

Per obtenir l'*authtoken,* els *tablets* realitzaran una petició al *MobileController* proporcionant el nom d'usuari i la contrasenya. Primer s'haurà de comprovar que aquestes credencials són correctes, aquesta verificació es durà a terme mitjançant el *UserService* que accedeix a la base de dades d'usuaris.

S'ha de tenir en compte que un *authtoken* només pot ser emprat per l'usuari que s'ha autenticat i que només serà vàlid durant un període de temps limitat. Per tant, és necessari que el servidor sàpiga a qui correspon cada *authtoken* i en quin moment ha estat generat (per poder invalidar-lo quan correspongui). Una primera opció per resoldre-ho consistiria en què el servidor guardés un llistat amb els *authtokens*, el moment de la seva generació i l'usuari al que

corresponen, aquestes dades haurien de ser persistents perquè els *authtokens* han de ser vàlids encara que el servidor es reiniciï. L'altra alternativa consisteix en incloure la informació necessària encriptada dins de l'*authtoken*.

Per evitar guardar més informació de la necessària a la base de dades i facilitar les validacions d'*authtokens,* aquests es generaran seguint els passos següents:

1. Es genera una cadena de caràcters aleatòria com la següent:

   3E10EB64-4062-487E-B42A-436E88E6CC54

2. Es crea una nova cadena de caràcters ajuntant la cadena aleatòria, el nom de l'usuari i la data i hora de generació (en milisegons) separats amb un caràcter "|". Per exemple:

   3E10EB64-4062-487E-B42A-436E88E6CC54|julia|1377630078513

3. S'encripta la cadena de caràcters creada al punt 2 i s'obté l'*authtoken*:

0DC13EC04E4AC210FFEED222FCE2256EDFCDB245ABD2DBBCCCBEC
B9FE2098EBD02A2149D7E5FC3371795CAD4E9D68CE916905D95AB80B
3F23DD57EC8D0C94D2BA550003A6B88F0DC

4.2.1.2. *Verificació d'*authtokens

Quan es realitzi una petició s'haurà d'acompanyar d'un *authtoken* i del nom d'usuari. El nom d'usuari és necessari per saber quines dades s'han de retornar. En cada petició s'haurà de verificar que l'*authtoken* és vàlid i que correspon a l'usuari que està fent la petició.

L'*AuthService* desencriptarà l'*authtoken* i n'extraurà el nom d'usuari, la data i l'hora de generació. L'*authtoken* serà considerat vàlid si el nom d'usuari que inclou coincideix amb el nom de l'usuari que fa la petició i no han passat més de 5 minuts des de la seva generació. En qualsevol altre cas, serà considerat invàlid i el *tablet* rebrà una resposta amb codi HTTP 401 *Unauthorized*. Llavors l'usuari haurà de tornar a autenticar-se per aconseguir un *authtoken* vàlid i repetir la petició.

4.2.2. Servei de notificacions *push* (*PushService*)

Tal com s'explica a l'apartat B.1.5, per a enviar notificacions *push*, el servidor envia els missatges al servidor GCM i aquest notifica als dispositius.

Els dispositius es registren al Servidor GCM, d'on obtenen el seu identificador, anomenat *RegID*, i l'envien al Servidor del CarismaTIC. Quan el servidor vol enviar una notificació *push* a un usuari, ha d'enviar el missatge al servidor GCM indicant el *RegID* del dispositiu al que va dirigit. Per tant, el servidor haurà de guardar una taula amb la relació d'usuaris i *RegID*s per poder enviar les notificacions.

Aquesta taula ha de ser persistent, perquè si es perdés la relació (en una caiguda del servidor, per exemple) els dispositius i el Servidor GCM considerarien que el registre encara és correcte i en realitat el Servidor no podria enviar notificacions perquè no sabria el *RegID* dels dispositius. Per fer la taula persistent s'ha creat el *PushRepository* (veure Fig. 29) que dóna accés a la base de dades on es guarden les relacions.

4.3. Gestió de les trucades

Al CarismaTIC es poden realitzar dos tipus de trucades: un metge/col·laborador pot trucar a un beneficiari i un beneficiari pot trucar a un altre beneficiari. Aquestes trucades s'han de tractar de manera diferent perquè els metges/col·laboradors accedeixen des d'un navegador web i els beneficiaris ho fan a través d'un *tablet.*

Per avisar de trucades entrants, de l'acceptació d'una trucada, de la cancel·lació d'una trucada, etc., el millor sistema són les notificacions *push*, que permeten enviar missatges als dispositius sense que aquests facin cap petició prèviament. Tanmateix, els navegadors no poden rebre notificacions *push*, i per tant, cal que facin *polling*. S'ha dissenyat un sistema que permet comunicar-se a través de notificacions *push* amb els *tablets* i que alhora deixa als navegadors fer *polling*. Tal com els casos d'ús indiquen, el metge/col·laborador no rebrà mai trucades sinó que serà el beneficiari l'únic que podrà rebre'n. Per tant, s'ha de dissenyar un sistema que permeti fer i rebre trucades des del tablet i fer trucades des de la web.

El sistema mantindrà un llistat amb les trucades actives i el seu estat. Quan els navegadors es trobin fent una trucada, estaran constantment fent *polling* al servidor, demanant cada segon l'estat actualitzat d'aquesta. Quan hi hagi un canvi en el seu estat, el sistema el reflectirà al llistat, de manera que el navegador, a la seva següent petició, rebrà el canvi. Si l'usuari al que afecta el canvi en la trucada és un beneficiari, se li enviarà una notificació *push*. D'aquesta manera s'assegura que quan hi hagi un canvi, tant els usuaris que accedeixen a través de *tablet* com els que ho fan via web seran avisats.

Les peticions relacionades amb trucades (que s'anomenaran *Rooms*) són gestionades pel *VideoconferenceController* (tant les del navegador com les del

tablet). El *VideoconferenceController* es comunica amb el *VideoconferenceService,* que és l'encarregat de mantenir el llistat amb les *Rooms* actives i d'actualitzar-ne l'estat quan correspon.

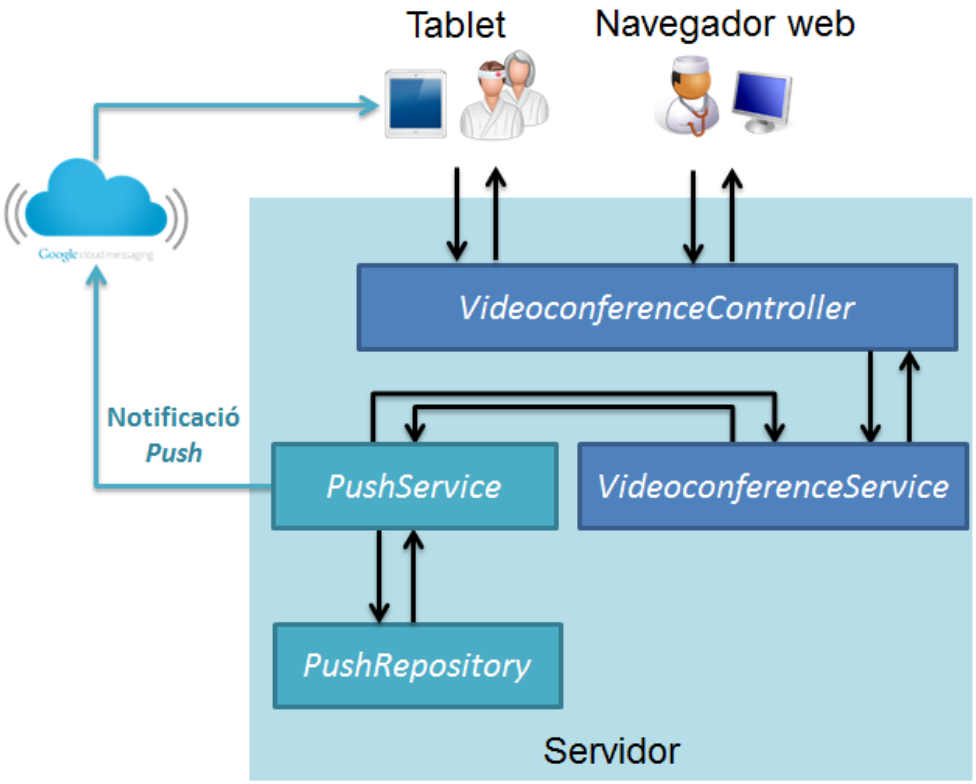

Fig. 30 Esquema de funcionament del mòdul de gestió de trucades del CarismaTIC.

El *VideoconferenceService* en funció del rol de l'emissor de la trucada es comportarà de manera diferent. Si l'emissor accedeix des de *tablet*, el servidor li enviarà una notificació *push* cada cop que hi hagi un canvi en l'estat de la *Room*. Per a això, emprarà el *PushService*, tal com s'ha explicat anteriorment.

En canvi, si l'emissor accedeix des del navegador, quan hi hagi un canvi en l'estat d'una trucada aquest es reflectirà al llistat de *Rooms* però no es notificarà a l'usuari, ja que es considera que aquest estarà fent polling i demanarà l'estat actualitzat de la trucada cada segon.

Per altra banda, com que el receptor de la trucada sempre serà un *tablet*, el servidor sempre podrà enviar-li notificacions *push*.

4.3.1. Estats de les trucades

Tota trucada s'identifica de manera única mitjançant un ID (*roomId*). A més, s'han establert set estats en què es pot trobar una *Room*:

- ***Connecting***: quan l'emissor ha iniciat la trucada però el receptor encara no ha confirmat que l'hagi rebuda.
- ***Calling***: quan el dispositiu del receptor notifica que ha rebut la trucada.
- ***Rejected***: quan el receptor rebutja la trucada.
- ***Accepted***: quan el receptor accepta la trucada.
- ***Canceled***: quan l'emissor cancel·la la trucada.
- ***Expired***: quan la trucada ha caducat. Quan es crea la trucada s'estableix un temps de vida de la mateixa, és a dir, el temps màxim que durarà la trucada si no es rep una resposta del receptor. Aquest temps el fixa el servidor a 40 s. Quan es crea una *Room*, es calcula en quin moment caducarà i s'afegeix a la notificació *push* un *Timestamp* que ho indica. D'aquesta manera, s'assegura que la trucada finalitzarà alhora en emissor i en receptor.
- ***Error***: quan el receptor no pot rebre trucades.

4.3.2. Trucades de la web al *tablet*

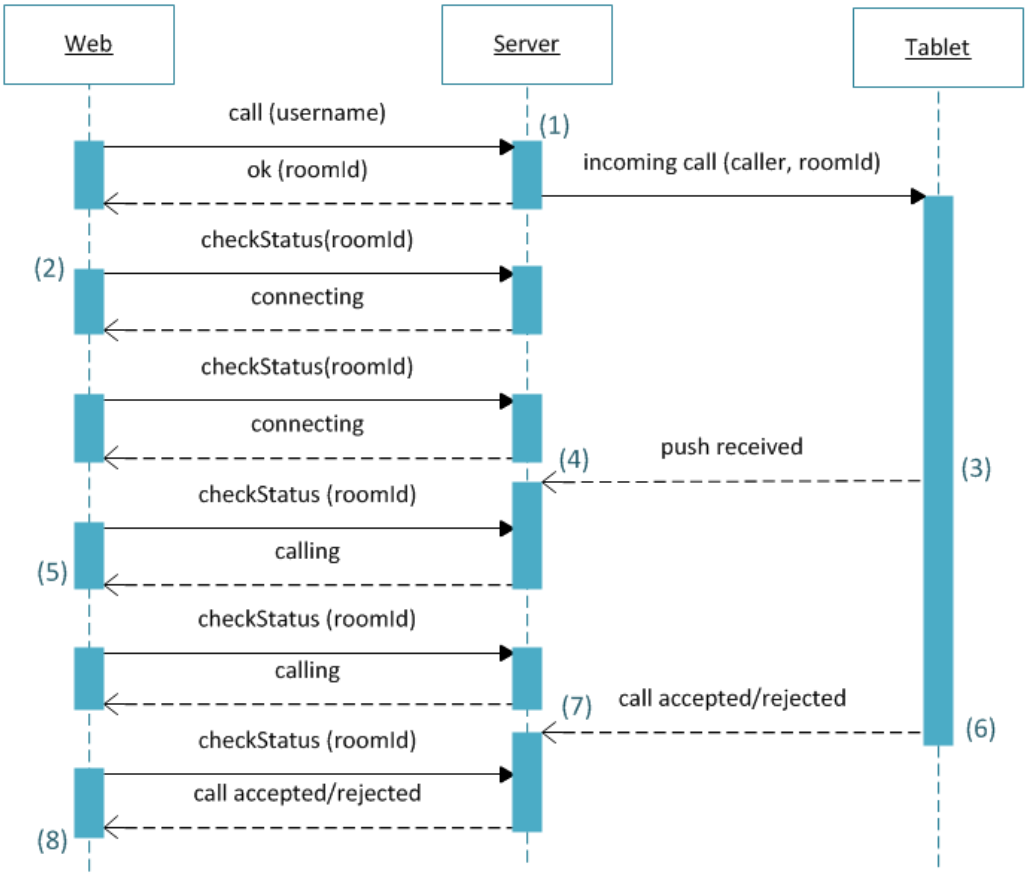

Fig. 31 Diagrama de seqüència que mostra els missatges que s'envien quan un metge o un col·laborador (web) truca a un beneficiari (*tablet*).

Les trucades entrants al *tablet* són gestionades pel Launcher, que és l'encarregat de mostrar la *CallingMeActivity* quan es rep la notificació. A la Fig. 31 es mostra el diagrama de seqüència d'una trucada que realitza un metge des de la web al *tablet* d'un beneficiari.

A continuació s'explica què passa en cada punt marcat a la Fig. 31:

(1) El servidor crea una *Room* (l'estat inicial de la qual és *connecting*) i en retorna l'ID a l'emissor perquè aquest pugui fer *polling* demanant-ne l'estat. Alhora envia una notificació *push* al dispositiu del beneficiari indicant-li que se l'està trucant. El navegador mostra a l'emissor el missatge de la Fig. 32 (a).

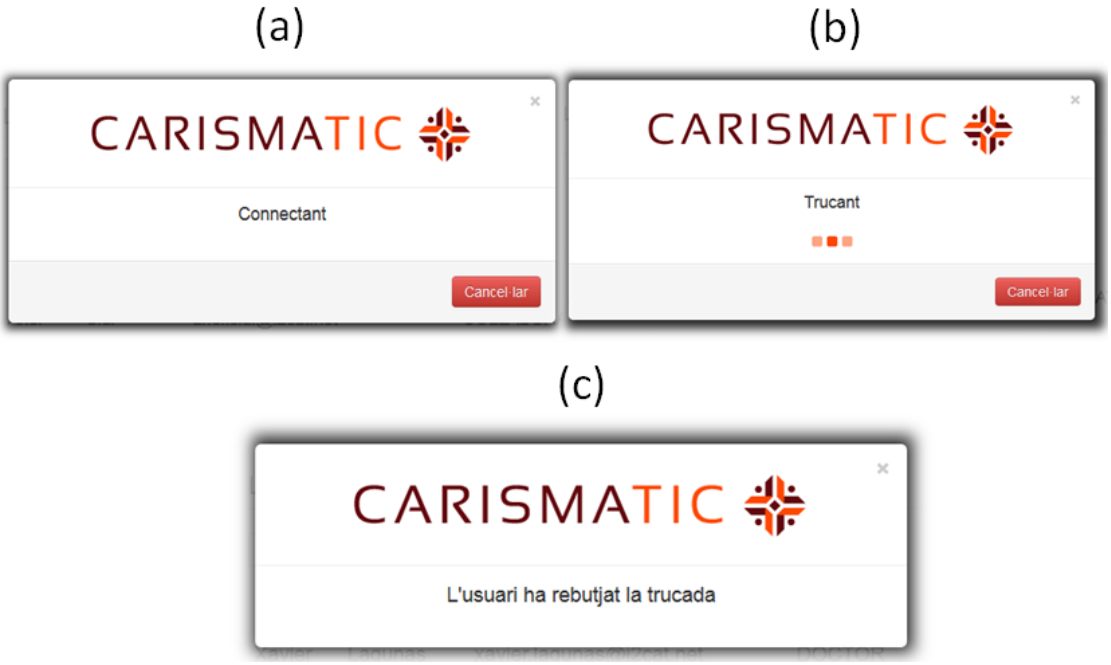

Fig. 32 Missatge que es mostra a la web mentre l'estat de la trucada és: (a) *connecting*. (b) *calling*. (c) *rejected*.

(2) El navegador comença a fer *polling* per tenir actualitzat l'estat de la trucada.
(3) El dispositiu del beneficiari notifica al servidor que ha rebut correctament la notificació *push* de trucada i que, per tant, està mostrant a l'usuari un avís com el de la Fig. 25.
(4) El servidor canvia l'estat de la trucada a *calling.*
(5) El navegador mostra a l'emissor que ja s'ha contactat amb el receptor i que se l'està trucant (veure Fig. 32 (b)).
(6) L'usuari accepta o rebutja la trucada: si l'accepta, s'obre l'aplicació de videoconferència; si la rebutja, es tanca la finestra d'avís. En ambdós casos s'informa de l'opció triada al servidor.

(7) El servidor canvia l'estat de la trucada a *accepted* o *rejected* segons correspongui.
(8) El navegador rep la resposta a la trucada: si la trucada ha estat acceptada, inicia la videoconferència; si ha estat rebutjada, informa a l'usuari tal com es pot veure a la Fig. 32 (c).

4.3.3. Trucades de *tablet* a *tablet*

Per altra banda, les trucades sortints del *tablet* són gestionades per l'aplicació de Contactes. Si està instal·lada l'aplicació de videoconferència, l'aplicació Contactes mostra l'opció de trucar a altres beneficiaris. A la Fig. 33 es mostra el diagrama de seqüència d'una trucada entre dos beneficiaris. Al *tablet* receptor, l'aplicació que gestiona les trucades és el Launcher, tal com s'ha explicat a l'apartat anterior.

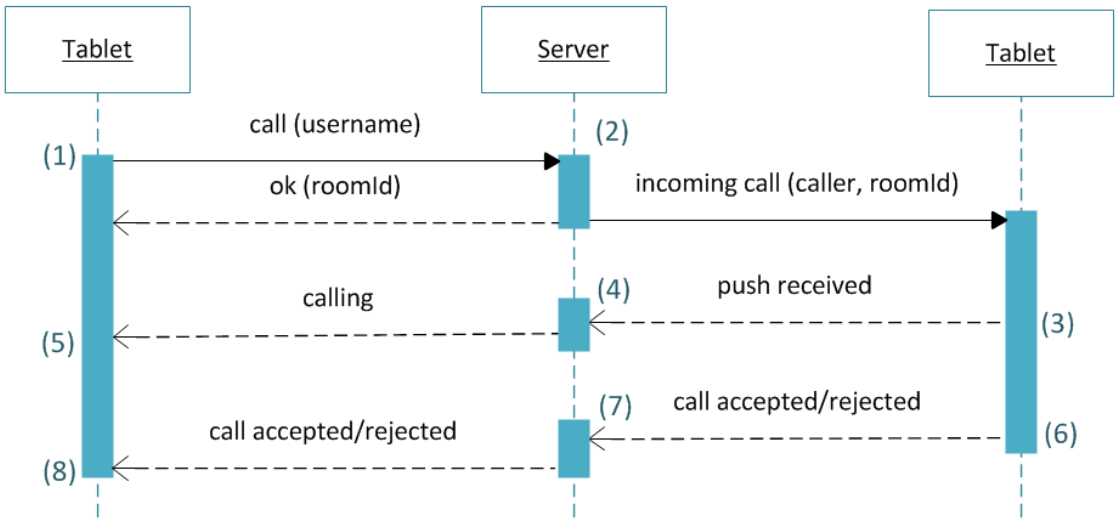

Fig. 33 Diagrama de seqüència que mostra els missatges que s'envien quan un beneficiari truca a un altre beneficiari.

Com es pot observar, ara hi ha moltes menys comunicacions, ja que els dos dispositius poden rebre notificacions *push* i no cal que cap de les dues parts faci *polling*. A continuació s'explica què passa en cada punt marcat a la Fig. 33:

(1) L'aplicació Contactes mostra a l'emissor un missatge per confirmar que vol fer la trucada (veure Fig. 34 (a)).

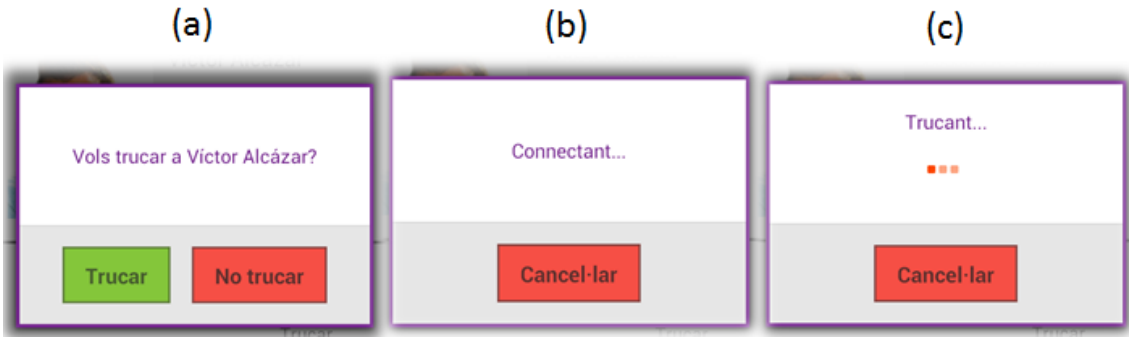

Fig. 34 Missatges que mostra l'aplicació de contactes durant el desenvolupament d'una trucada (a) Inicialment, permetent a l'usuari confirmar que vol realitzar la trucada. (b) Quan s'ha iniciat una trucada però encara no s'ha rebut la notificació *push* indicant que el seu estat ha passat a *calling*. (c) Quan s'ha rebut una notificació indicant que el receptor ja ha rebut la trucada.

(2) El servidor crea una *Room* (l'estat inicial de la qual és *connecting*) i en retorna l'ID a l'emissor. El *tablet* de l'emissor mostra el missatge de la Fig. 34 (b). Alhora, el servidor envia una notificació *push* al *tablet* receptor indicant-li que se l'està trucant.
(3) El dispositiu del receptor notifica al servidor que ha rebut correctament la notificació *push* de trucada i que, per tant, està mostrant a l'usuari l'avís corresponent (veure Fig. 25).
(4) El servidor canvia l'estat de la trucada a *calling* i envia una notificació *push* a l'emissor per informar-lo.
(5) El tablet mostra a l'emissor que ja s'ha contactat amb el receptor i que se l'està trucant (Fig. 34 (c)).
(6) L'usuari accepta o rebutja la trucada: si l'accepta, s'obre l'aplicació de videoconferència; si la rebutja, es tanca la finestra d'avís. En ambdós casos s'informa de l'opció triada al servidor.
(7) El servidor canvia l'estat de la trucada a *accepted* o *rejected* segons correspongui.
(8) El *tablet* emissor rep la resposta a la trucada: si la trucada ha estat acceptada, inicia la videoconferència; si ha estat rebutjada, ho indica a l'usuari i tanca la finestra.

4.3.4. Cancel·lació d'una trucada

Quan l'emissor d'una trucada la cancel·la, ho notifica al servidor, que canvia l'estat a *canceled*, i ho notifica mitjançant una notificació *push* al dispositiu corresponent. S'ha de tenir en compte que només pot rebre trucades un beneficiari i que, per tant, només pot rebre la cancel·lació d'una trucada un *tablet*.

4.3.5. Expiració d'una trucada

La notificació *push* que indica a un *tablet* receptor que té una trucada entrant, també li indica en quin moment expira aquesta. El que farà és mostrar el missatge de trucada entrant (veure Fig. 25) mentre no s'arribi al moment d'expiració. Si arriba aquest moment i l'usuari no ha contestat la trucada (acceptant-la o rebutjant-la) es deixa de mostrar l'avís a l'usuari.

4.3.6. Error en una trucada

Es possible que un usuari truqui a un beneficiari que no tingui l'aplicació de videoconferència instal·lada. En aquest cas, el *tablet* mostrarà un missatge d'error a l'usuari i ho notificarà al servidor, que posarà la trucada en estat d'error i notificarà a l'altre usuari, si es tracta d'un beneficiari.

CAPÍTOL 5. INTEGRACIÓ D'APLICACIONS AL LAUNCHER

Com ja s'ha vist, l'objectiu del Launcher és que diverses aplicacions s'hi puguin integrar, aprofitant les funcionalitats que aquest aporta. Per tant, el Launcher no és completament independent de les aplicacions, sinó que serveix per fer-les més intel·ligents. En aquest capítol s'explicarà en què consisteix la integració d'una aplicació al Launcher mitjançant un exemple: l'aplicació Contactes.

5.1. Funcions de l'aplicació Contactes

L'aplicació Contactes mostra a l'usuari el seu llistat de contactes i li dóna l'opció de trucar-los i d'enviar-los missatges. Tant les trucades com l'enviament de missatges es durà a terme en dues aplicacions independents, la de videoconferència i la de missatgeria. L'aplicació Contactes haurà d'executar aquestes aplicacions quan sigui necessari.

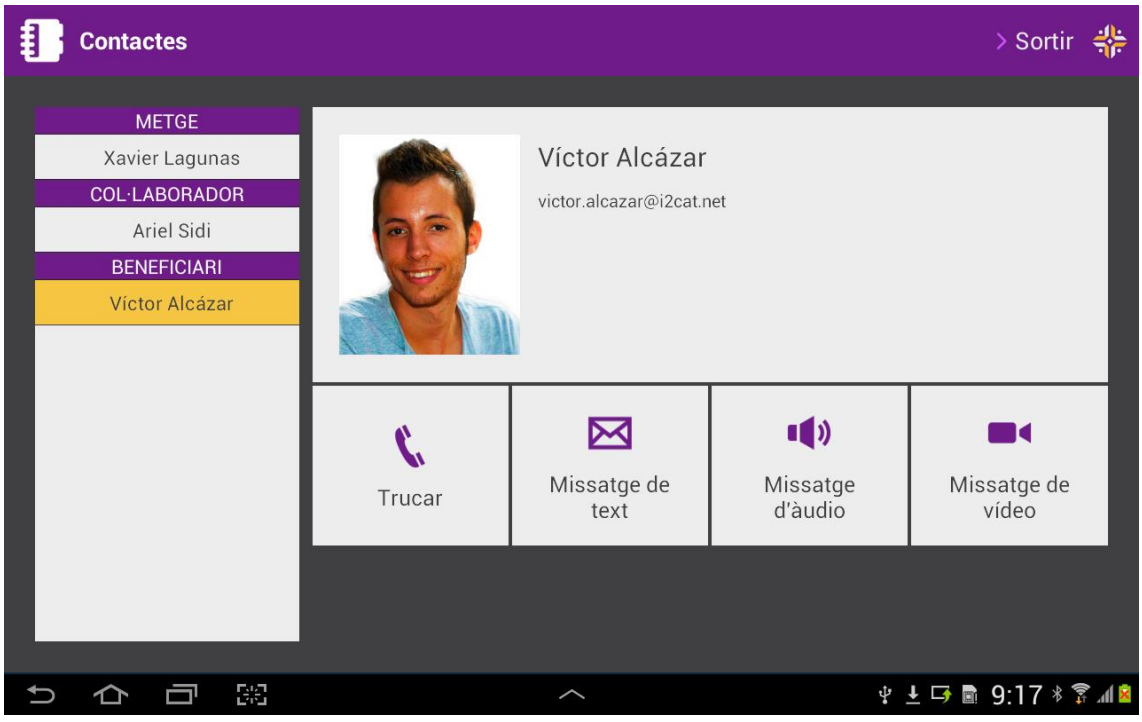

Fig. 35 Captura de l'aplicació de contactes en la qual es veu que l'usuari, a més de veure les dades del contacte, pot trucar-lo i enviar-li missatges.

Tal com s'explica a l'apartat 4.3.3, la gestió de les trucades sortints l'ha de realitzar l'aplicació Contactes. La seva feina és notificar al servidor que es vol fer una trucada, rebre les notificacions amb la resposta i obrir l'aplicació de videoconferència si la trucada és acceptada. A la Fig. 35 es pot veure una captura de l'aplicació en què el contacte és un beneficiari, se li pot trucar i enviar missatges de tres tipus: text, àudio i vídeo. Si, en lloc de ser un

beneficiari, es tractés d'un col·laborador o d'un metge, només es mostrarien les opcions de missatgeria, ja que un beneficiari només pot trucar a altres beneficiaris.

5.2. Integració al Launcher

L'aplicació Contactes, per poder mostrar a l'usuari el llistat de contactes, prèviament haurà de descarregar-los del servidor. Per a fer-ho necessitarà autenticar-se mitjançant nom d'usuari i contrasenya i obtenir així un *authtoken* que li permeti autenticar les peticions. Si es tractés d'una aplicació qualsevol, hauria de demanar a l'usuari les seves credencials per poder realitzar l'autenticació. Però com que es tracta d'una aplicació integrada al Launcher, aquest aporta eines que li permeten aconseguir l'*authtoken* sense necessitat que l'usuari introdueixi les credencials novament.

Abans de demanar l'*authtoken*, haurà de comprovar si hi ha un compte del CarismaTIC configurat, per la qual cosa disposa de les funcions proporcionades pel *LauncherUtilities*, que és la classe que proporciona totes les funcions necessàries perquè una aplicació s'integri al Launcher.

Tal com es veu a la Fig. 22 en color lila, les aplicacions que s'integrin al Launcher només hauran de demanar a l'*AccountManager* d'Android que els proporcioni l'*authtoken* i ja podran autenticar les peticions. Un cop aquest *authtoken* deixi de ser vàlid només caldrà que ho notifiquin a l'*AccountManager* i aquest s'encarregarà d'obtenir-ne un de vàlid. S'ha de destacar que, perquè una aplicació pugui aconseguir l'*authtoken* del compte del CarismaTIC, ha d'haver estat firmada per la mateixa entitat que l'*Authenticator*, és a dir, i2CAT. Així s'assegura que només les aplicacions d'i2CAT podran aconseguir els *authtokens*. Si es volgués que una aplicació d'un tercer accedís als *authtokens* del CarismaTIC s'hauria d'afegir al Launcher un sistema que permetés donar permisos a altres aplicacions per fer servir les credencials, i aquestes aplicacions haurien de demanar permís a l'usuari per fer-les servir.

Per altra banda, per a la gestió de les trucades sortints serà necessari que l'aplicació Contactes pugui rebre les notificacions *push* (indicant que l'altre usuari ja ha rebut la trucada, que s'ha acceptat la trucada, etc.). Aquestes notificacions són gestionades al *PushUtilities* que, com s'ha explicat a l'apartat 3.4.3, decidirà com les tracta en funció de la seva prioritat. Com que el *PushUtilities* enviarà en b*roadcast* les notificacions relacionades amb les trucades, només cal que l'aplicació Contactes tingui un *BroadcastReceiver* que detecti quan són enviades.

5.3. Funcionament de l'aplicació Contactes

Al diagrama de flux de la

Fig. 36 es mostra el procés d'inici de l'aplicació. El primer que fa a l'iniciar-se és comprovar si hi ha un compte configurat i si no el troba mostra un error a l'usuari i tanca l'aplicació.

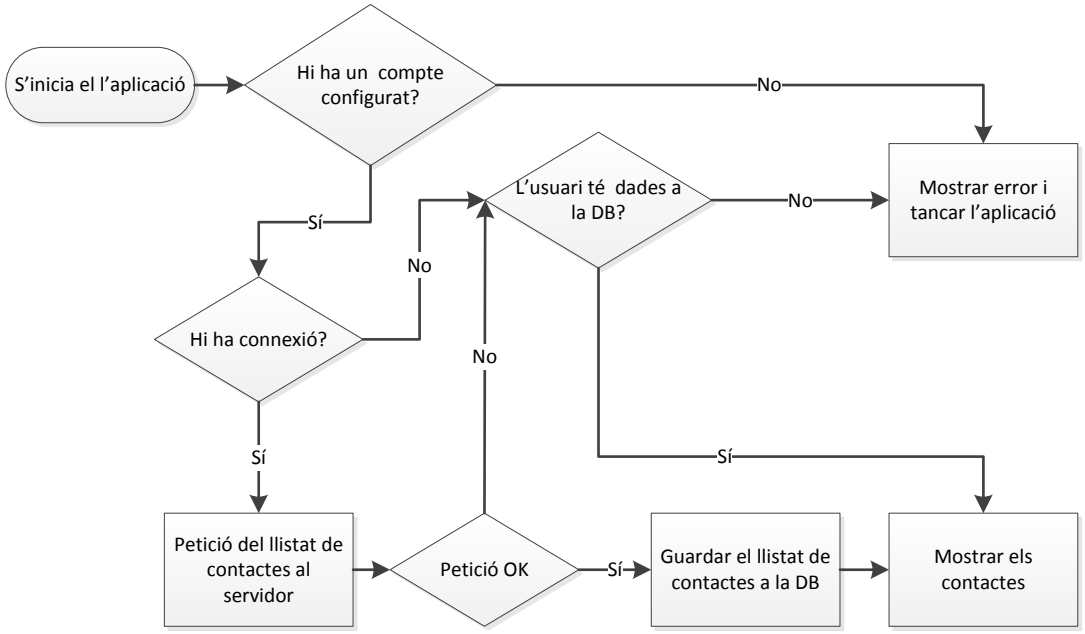

Fig. 36 Diagrama de flux del procés d'inici de l'aplicació de contactes.

Si troba el compte, comprova si hi ha connexió, i en aquest cas fa la petició de contactes al servidor. Si aquesta petició va bé, guarda els contactes a la base de dades i els mostra a l'usuari. En cas que no tingués connexió, consultaria la base de dades, i si aquesta tingués contactes els mostraria. Si no hi hagués connexió ni contactes a la base de dades, mostraria un error a l'usuari i tancaria l'aplicació.

CAPÍTOL 6. CONCLUSIONS

6.1. Avaluació dels requeriments

En començar aquest projecte es van definir una sèrie de requeriments. A continuació s'explicarà com s'ha verificat el seu compliment i els problemes que s'han trobat.

Durant el desenvolupament es va trobar un inconvenient a l'hora de complir el requeriment funcional de descarregar i instal·lar qualsevol aplicació del llistat que no es trobés instal·lada al dispositiu. Com que Android, per motius de seguretat, no permet instal·lar una aplicació al dispositiu sense l'autorització explícita de l'usuari, no s'ha pogut complir aquest requeriment. La solució més aproximada que es va trobar consisteix en facilitar al màxim la instal·lació a l'usuari interaccionant amb ell només quan és estrictament necessari per a autoritzar la instal·lació. Pel que fa a la resta de requeriments funcionals, s'ha fet un conjunt de proves per verificar que es compleixen i el resultat ha estat satisfactori.

A continuació s'analitza el compliment de cada un dels requeriments no funcionals:

- **Seguretat**: totes les connexions amb el servidor van xifrades mitjançant HTTPS, a més d'incorporar l'*authtoken* que les autentica.
- **Usabilitat**: tal com s'ha vist a les diferents captures, tots els botons són grans i van acompanyats de text per aconseguir que siguin el més intuïtius possible (exceptuant el botó que obre el menú d'opcions, que és petit per evitar que sigui premut per error). Quan es produeix un error, es mostra a l'usuari la informació en un *toast* del CarismaTIC, que es manté en pantalla més estona que un *toast* normal. Per acabar, el que fa més usable el Launcher és que automatitza la majoria d'accions. Tal com s'ha vist el Launcher realitza moltes activitats de manera automàtica quan s'inicia i només demana la interacció amb l'usuari quan és estrictament necessària.
- **Estabilitat**: el Launcher detecta tots els errors i no es penja. Quan es produeix un error es mostra la informació a l'usuari mitjançant un *toast* del CarismaTIC.
- **Portabilitat**: el Launcher funciona en dispositius amb Android 4.0.3 o superior. Tal com es pot veure a l'ANNEX C, adapta les seves vistes en funció de si la pantalla és de 7'' o de 10''.
- **Multi-idioma**: el Launcher detecta l'idioma del dispositiu i configura els seus textos en conseqüència. Si l'idioma del *tablet* no és ni català ni castellà, configura els textos en català. Si es volgués afegir un tercer idioma només caldria crear un nou arxiu strings.xml amb els textos traduïts.
- **Disponibilitat**: el Launcher seguirà funcionant sense connexió. Moltes aplicacions no tenen sentit sense connexió a internet, però es podrà accedir a les aplicacions que ja estan instal·lades.

6.2. Objectius assolits

L'objectiu principal d'aquest TFG era la creació d'un Launcher Android que fos usable per a gent gran. Aquest objectiu ha repercutit en tot el desenvolupament de manera significativa, ja que implica que cal automatitzar tots els processos i minimitzar la interacció amb l'usuari. Per tant, l'aplicació ha de ser prou intel·ligent per prendre decisions sense involucrar a l'usuari si no és estrictament necessari. Un altra part molt important de la usabilitat de l'aplicació té a veure amb el disseny: s'ha creat una interfície específica pel CarismaTIC que adapta el dispositiu a persones grans i alhora en manté totes les funcionalitats, permetent que altres usuaris puguin fer servir el dispositiu per realitzar activitats que no tinguin a veure amb el CarismaTIC.

Un altre objectiu molt important era aconseguir que el Launcher es pogués gestionar remotament. El fet que les aplicacions que es mostren puguin canviar en qualsevol moment per decisió del metge implica que el dispositiu ha d'estar en contacte permanent amb el servidor. Per aconseguir-ho s'han emprat notificacions *push* que han permès comunicar l'aplicació amb el servidor de manera eficient.

Per acabar, l'últim objectiu consistia en dissenyar un protocol de comunicació i implementar-lo. S'han creat una sèrie de *Services* i *Controllers* que s'encarreguen de la gestió de les peticions i de l'enviament de les notificacions *push*.

6.3. Impacte mediambiental

Podria semblar que aquest projecte no té un impacte mediambiental gaire important. Tanmateix, a continuació es veurà que si que té un efecte sobre l'estalvi energètic, que és un element molt important en el manteniment del medi ambient.

Gràcies al CarismaTIC, pacients que abans havien de traslladar-se a la consulta del seu metge per poder-li fer consultes, ara ho poden fer mitjançant videoconferència. A més, els pacients (o els seus cuidadors) poden introduir les dades biomèdiques a la plataforma sense necessitat que un col·laborador vagi a casa seva. Així s'estalvien molts trasllats en vehicles que consumeixen combustibles fòssils.

6.4. Conclusions personals

La realització d'aquest projecte m'ha aportat molts coneixements i experiència en programació d'aplicacions Android. Ja havia fet aplicacions per a Android abans, però en general es tractava d'aplicacions en HTML5 i, per tant, no tenia gaire experiència en programació nativa. L'aplicació m'ha demanat molt d'esforç, sobretot en les parts més complexes com la gestió de comptes o el disseny del protocol de comunicació. Abans d'aquest projecte, no tenia

coneixements de disseny, i després de la seva realització puc dir que tinc clars els conceptes més bàsics per a poder dissenyar una interfície d'usuari tenint en compte diversos factors.

Per altra banda, com que el Launcher forma part del CarismaTIC, m'he hagut de coordinar amb un equip per dur a terme la feina. A l'inici del projecte vaig participar en reunions de descripció de requeriments, i un cop ja estava iniciat he assistit a diverses reunions de seguiment. Per organitzar la feina hem seguint la metodologia de treball SCRUM, i jo m'he encarregat de planificar les meves tasques i d'estimar el temps necessari per dur-les a terme.

Molts dels coneixements adquirits durant la carrera m'han servit en la realització d'aquest projecte. El més útil ha estat el treball en equip; durant tota la titulació hem realitzat projectes en grup i això m'ha permès familiaritzar-me amb eines i procediments que permeten organitzar-te i assolir els objectius del projecte. Per altra banda, els coneixements d'Android adquirits a la carrera i a les pràctiques d'empresa m'han permès arribar a un nivell de complexitat de l'aplicació que d'una altra manera no hagués aconseguit.

6.5. Treballs futurs

Són moltes les possibilitats que dóna aquest projecte per a ser ampliat en un futur. Per començar, és possible anar agregant-hi funcionalitats creant aplicacions que s'integrin al Launcher.

Per altra banda, es podrien fer una sèrie de canvis al Launcher i d'aquesta manera fer-lo compatible per altres tipus d'usuaris. Per exemple, es podria afegir una funcionalitat de *text-to-speech* que llegís a l'usuari tots els textos de la pantalla. D'aquesta manera es podria ampliar el públic a usuaris amb dificultats visuals.

CAPÍTOL 7. REFERÈNCIES

[1] «Bouncy Castle,» [En línia]. Available: http://www.bouncycastle.org/. [Últim accés: juliol 2013].

[2] «Android Developers,» [En línia]. Available: http://developer.android.com/. [Últim accés: setembre 2013].

[3] «Google Cloud Messaging,» [En línia]. Available: http://developer.android.com/google/gcm/index.html. [Últim accés: juny 2013].

[4] «DB4O,» [En línia]. Available: http://www.db4o.com/. [Últim accés: maig 2013].

[5] «Maven Repository,» [En línia]. Available: http://mvnrepository.com/. [Últim accés: juny 2013].

[6] «Apache Maven,» [En línia]. Available: http://maven.apache.org/. [Últim accés: abril 2013].

[7] «Roboguice,» [En línia]. Available: https://github.com/roboguice/roboguice. [Últim accés: agost 2013].

CAPÍTOL 8. GLOSSARI

APK: format d'arxiu que es fa servir per distribuir les aplicacions Android.

Authtoken: cadena de caràcters que autentica una petició d'un usuari davant del servidor.

Backoffice: servei ofert per una plataforma perquè se la pugui gestionar. Els únics usuaris que hi tenen accés són els administradors.

***Broadcast*:** difusió d'un missatge de manera que pugui arribar a tots els possibles receptors.

Controller: component de *Spring* que s'encarrega de la gestió de les peticions. Per poder atendre-les ha d'accedir als *Services*.

Framework: plataforma de *software* que permet abstraure el programador de certes funcions.

GCM: *Google Cloud Messaging,* servei que ofereix Google que permet emprar notificacions *push* a la teva aplicació.

HTTP: protocol de comunicació que es fa servir en les transaccions de la *World Wide Web*. En totes les respostes d'una comunicació HTTP hi ha un codi que indica si la petició ha anat bé i, si hi ha hagut un error, quin ha estat.

HTTPS: protocol HTTP afegint seguretat mitjançant el protocol SSL o TLS. Permet assegurar a l'usuari que s'està comunicant amb qui el seu interlocutor diu ser. Per aconseguir-ho fa falta un certificat que hagi estat firmat per una tercera entitat de confiança.

ID: cadena de caràcters que identifica de forma única un element de qualsevol sistema.

Plug-in: complement que permet ampliar les funcionalitats d'un *software*.

***Polling*:** sistema de comunicació que consisteix en fer peticions de manera continuada per mantenir actualitzat algun element (marcador d'un partit de futbol, per exemple).

Query: cadena de caràcters que representa una consulta en una base de dades relacional.

Repository*:* component de Spring que permet accedir a la base de dades.

SCRUM: conjunt de tècniques per gestionar i controlar el desenvolupament de software de manera àgil.

Service: component de *Spring* que aporta la lògica a la plataforma. Es comunica amb els *Repositories* per accedir a la base de dades.

SQLite: sistema de gestió de bases de dades relacionals.

Text-to-speech: sistema que converteix el text en veu sintètica.

Thread**:** fil d'execució d'una aplicació. Un programa pot tenir múltiples *threads* executant-se paral·lelament.

Timestamp**:** marca que indica la data i l'hora en la que s'ha generat un element.

URI: adreça que permet localitzar un recurs.

URL: adreça en què es pot localitzar un recurs web.

ANNEXOS

ANNEX A DIAGRAMES DE CASOS D'ÚS

Aquest annex recull els diagrames de casos d'ús dels diferents actors del CarismaTIC. Aquests estan dividits en diferents mòduls per facilitar-ne la comprensió.

A.1 Diagrama de casos d'ús de l'administrador

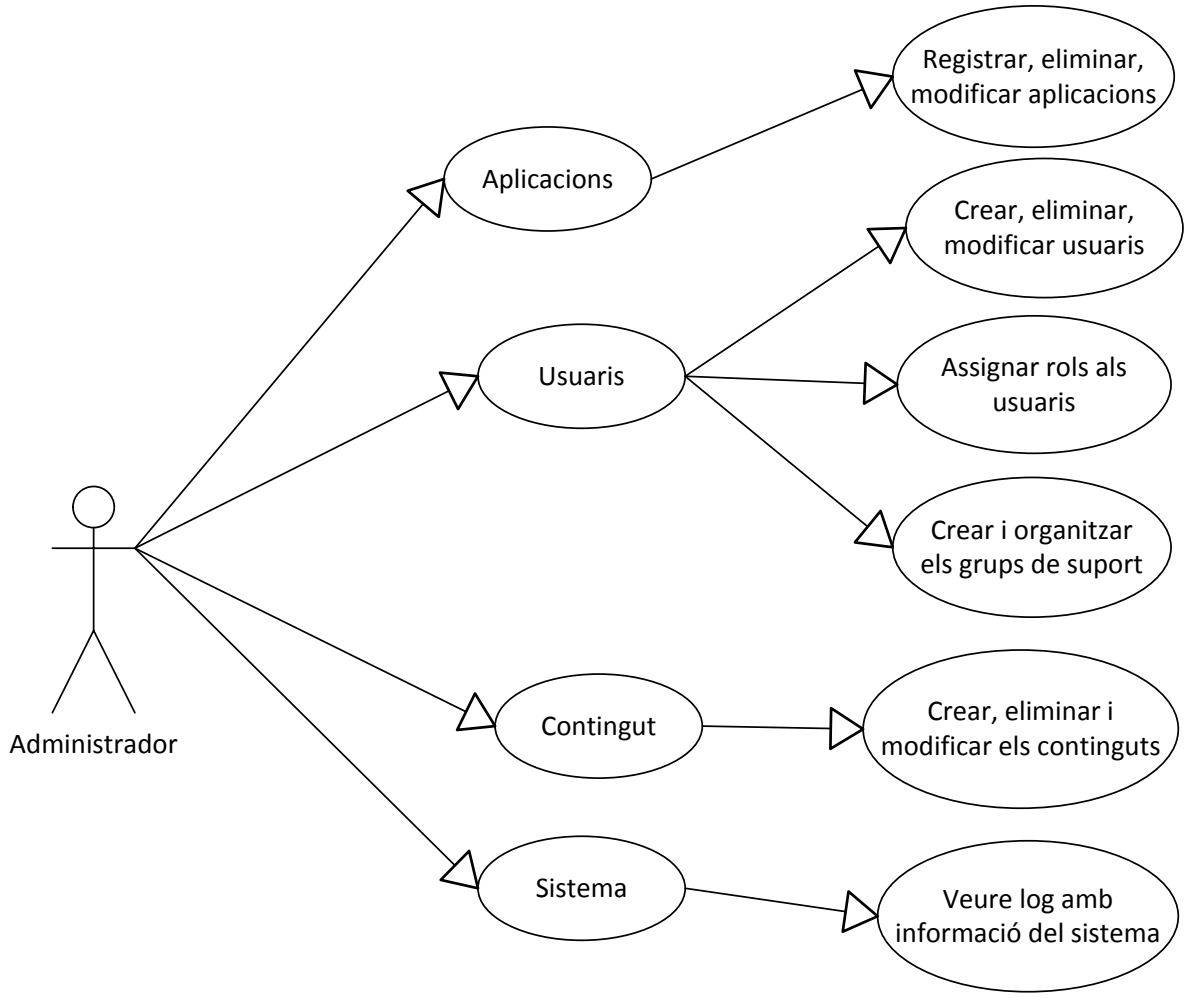

Fig. 37 Model de casos d'ús de l'administrador.

A.2 Diagrama de casos d'ús del metge

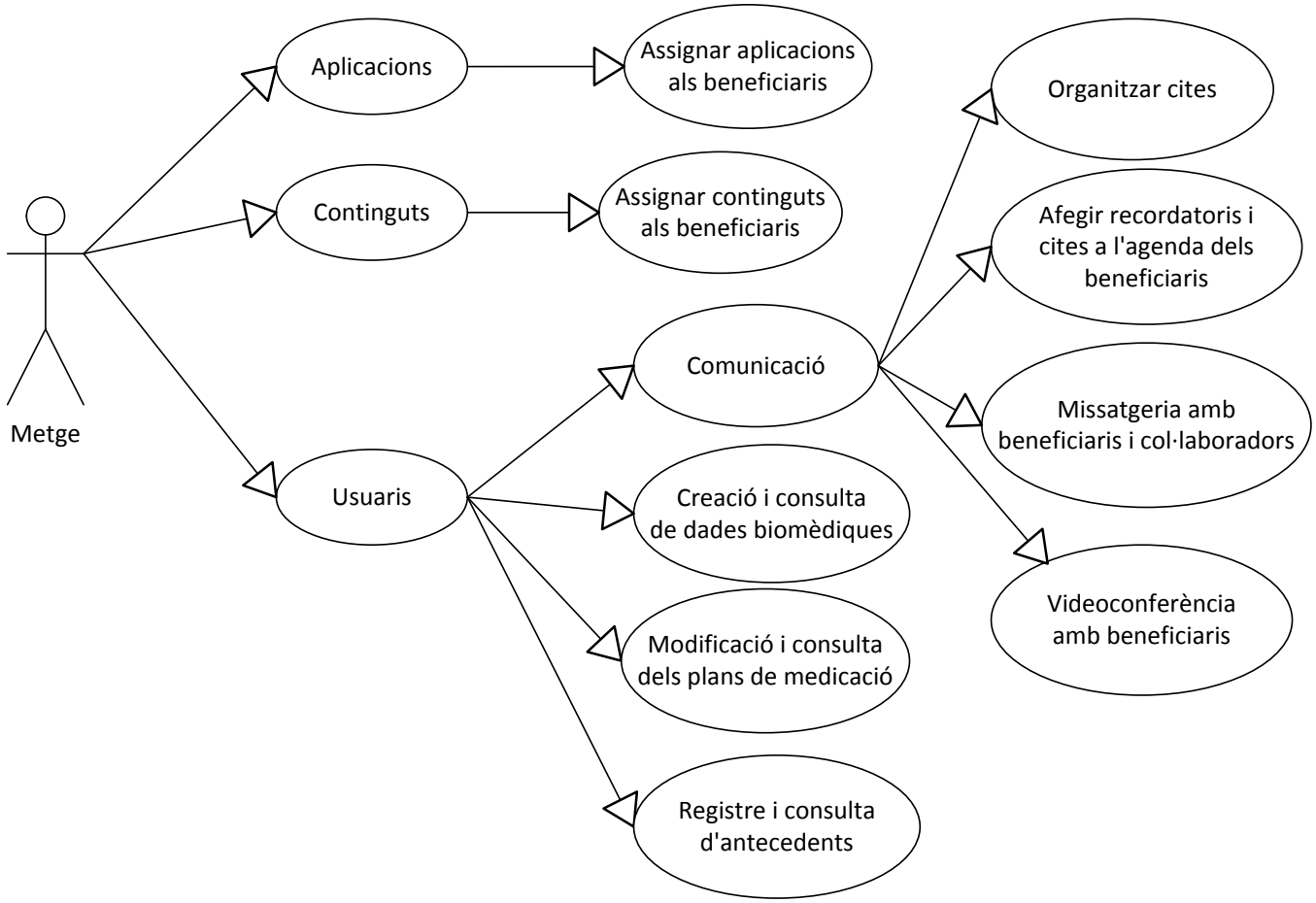

Fig. 38 Model de casos d'ús del metge.

A.3 Diagrama de casos d'ús del col·laborador

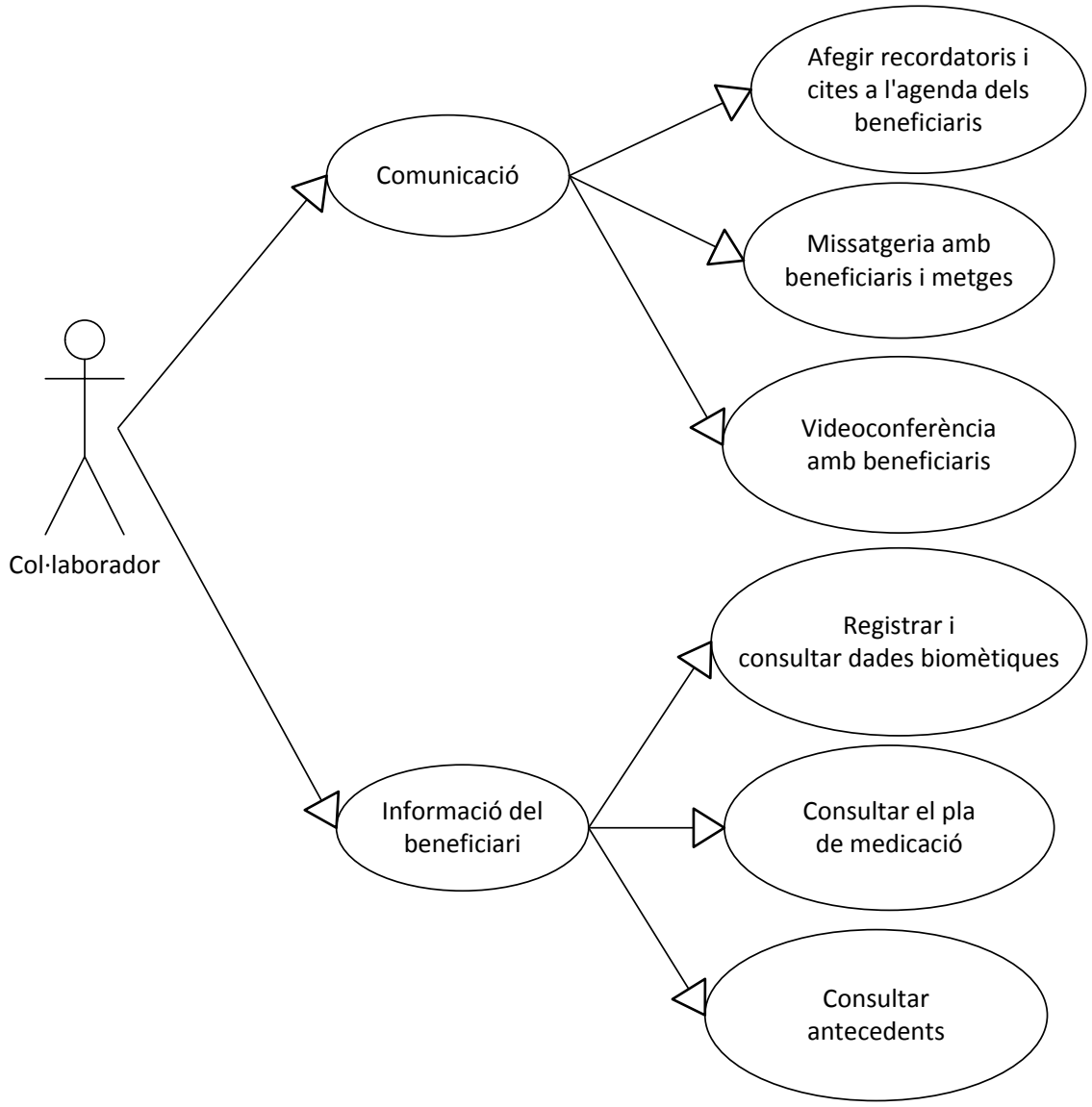

Fig. 39 Model de casos d'ús del col·laborador.

A.4 Diagrama de casos d'ús del beneficiari

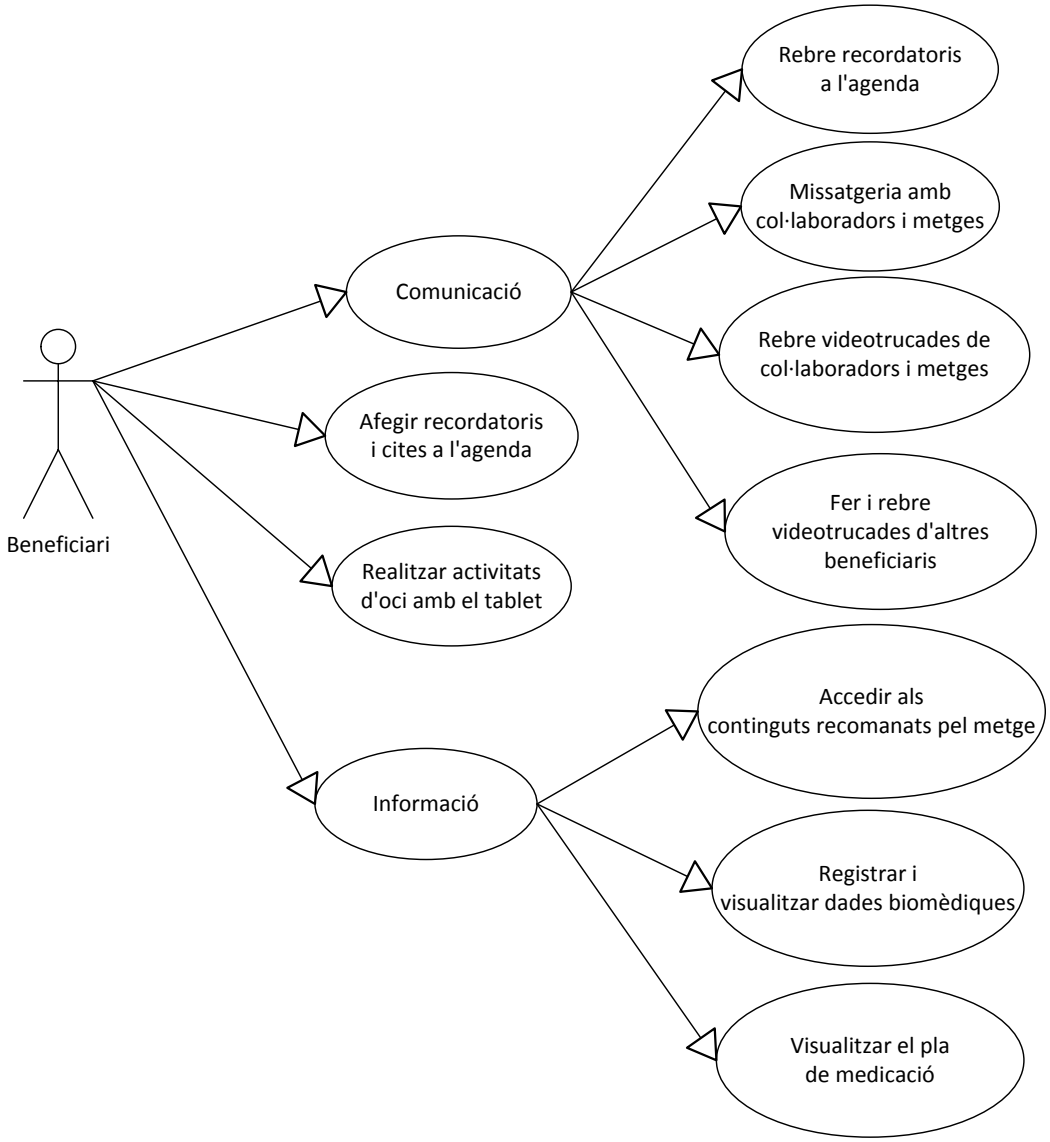

Fig. 40 Model de casos d'ús del beneficiari.

ANNEX B TECNOLOGIES

En aquest annex es descriuran les diferents tecnologies que s'han emprat per al desenvolupament del Launcher. L'objectiu és exposar què han aportat aquestes tecnologies al projecte; no es pretén fer-ne una descripció exhaustiva.

B.1 Android

Android és un sistema operatiu per a dispositius mòbils basat en Linux i dissenyat principalment per a dispositius que tenen pantalla tàctil. Hi ha diversos llenguatges que es poden emprar per desenvolupar aplicacions per a Android: C, C++, Java, HTML5... En aquest projecte s'ha emprat Java, que és un llenguatge de programació orientat a objectes.

Als apartats següents es descriuran els components que formen les aplicacions Android i els diferents recursos visuals, textuals, etc. que es poden agregar a una aplicació.

B.1.1 Components

Una aplicació Android consta de diferents components, que poden ser dels següents tipus:

- ***Activity***: cada una sol representar una vista de l'aplicació amb la qual pot interactuar l'usuari.
- ***Service***: s'executa en *background,* és a dir, darrere de la resta d'aplicacions sense que l'usuari se n'adoni. No disposa d'interfície d'usuari i acostuma a realitzar tasques que cal que estiguin executant-se durant una llarga estona o un temps indefinit, com per exemple la recepció de notificacions.
- ***ContentProvider****:* s'encarrega de gestionar les dades de l'aplicació (en bases de dades SQLite, per exemple) i permet accedir a altres dades del dispositiu com el llistat de contactes de l'usuari, els arxius guardats a la memòria, etc.
- ***BroadcastReceiver****:* respon als missatges difosos pel sistema o per una aplicació (*broadcasts*). Per exemple, quan canvia l'orientació del dispositiu de vertical a horitzontal el sistema difon un *broadcast* que serà rebut pels *BroadcastReceivers* que estiguin interessats en els canvis d'orientació. Una aplicació pot difondre *broadcasts* de manera que puguin ser rebuts per altres aplicacions o per altres components de la mateixa aplicació.

Els components que formen l'aplicació han de ser declarats a l'arxiu *AndroidManifest.xml*. En aquest arxiu també es defineixen els permisos que tindrà l'aplicació; per exemple, per a poder fer servir la càmera del dispositiu cal especificar-ho a l'*AndroidManifest.xml*. D'aquesta manera, en el moment de la instal·lació, l'usuari serà informat que l'aplicació farà servir la càmera i podrà decidir si la instal·la o no.

A l'*AndroidManifest.xml* també cal definir la versió del sistema operatiu Android per la que es programarà. Això es fa mitjançant el camp *minSdkVersion* que defineix la versió mínima d'Android que haurà de tenir un dispositiu per a poder executar l'aplicació.

B.1.2 Recursos

Una aplicació Android pot disposar de diferents tipus de recursos. Els més rellevants per aquest projecte són:

- ***Drawable***: recursos visuals, que poden ser imatges o arxius XML que descriguin imatges. Per poder suportar pantalles amb diferents densitats de píxels, per a cada densitat de píxels suportada es defineix una carpeta (amb la corresponent imatge). El sistema operatiu reconeix automàticament la carpeta adient segons la densitat de píxels del dispositiu.
- ***Strings***: arxius XML que contenen les cadenes de caràcters que es fan servir a l'aplicació per mostrar informació a l'usuari. Hi pot haver un arxiu Strings.xml per a cada llengua de manera que el dispositiu pugui triar l'arxiu que es correspon a l'idioma configurat per l'usuari.
- ***Layout***: arxius XML que descriuen les diferents vistes que podran mostrar les *activities*. Es poden definir vistes diferents en funció de la mida del dispositiu i de l'orientació d'aquest.
- ***Style****:* arxius XML que contenen els estils de l'aplicació, de manera que es pugui donar format als elements visuals d'una forma ràpida. Per exemple, si l'aplicació fa servir sovint botons de color vermell amb lletra de color blanc, es pot crear un estil que s'anomeni "RedButton" de manera que només calgui assignar-lo a un element perquè es configuri d'aquesta manera.

Per a més informació sobre components i recursos d'Android, es pot consultar [2].

B.1.3 Managers

Android proporciona un conjunt de classes anomenades *Managers* que donen accés a diferents serveis del sistema operatiu. El *LocationManager* permet accedir a les dades de geolocalització del dispositiu, el *NotificationManager* permet afegir missatges al centre de notificacions i mostrar avisos, etc.

Al projecte es fan servir dos *Managers*: l'*AccountManager* i el *PackageManager*. El primer serveix per a gestionar els comptes d'usuari i el segon, per a accedir a la informació de les aplicacions instal·lades al dispositiu.

B.1.4 Distribució d'aplicacions

Android disposa d'un sistema propi de distribució d'aplicacions: el *Google Play Store.* Es tracta d'una botiga d'aplicacions a la que es pot accedir des de qualsevol dispositiu Android que permet buscar i instal·lar aplicacions.

Per identificar una aplicació de manera única al *Play Store* es fa servir el camp *PackageName*, que es defineix a l'*AndroidManifest.xml* de l'aplicació. Per assegurar que aquest valor és únic, es forma amb l'URL de l'organització en ordre invers i el nom de l'aplicació. Per exemple, el *PackageName* del Launcher és *net.i2cat.csade.launcher* i el de l'aplicació de contactes, *net.i2cat.csade.contacts*.

Per tant, per a distribuir una aplicació només cal configurar-li un *PackageName* únic i pujar-la al *Play Store*. El format en què es distribueix una aplicació és l'APK, que és el resultat de compilar-la i empaquetar-la. Així doncs, aquest és el tipus d'arxiu que permet instal·lar una aplicació Android.

B.1.5 Notificacions *push*

En les comunicacions habituals client–servidor, les peticions s'originen als clients i els servidors les contesten, sistema que es coneix amb el nom de *pull*. En el sistema *push,* pel contrari*,* és el servidor qui inicia la comunicació amb el client.

Quan és necessari actualitzar la informació al client molt sovint (per a actualitzar un llistat de contactes, un marcador d'un partit de futbol, etc.) el sistema *pull* ha d'estar constantment fent peticions. Si es tracta d'un dispositiu amb connexió ADSL i connectat a la corrent alterna, això no és un problema. En canvi, quan el dispositiu s'alimenta amb bateria i, a més, és molt possible que tingui una limitació en quant a capacitat descarregada; el sistema *pull* es converteix en un problema pel seu ús intensiu de la xarxa. Per això, en els dispositius mòbils s'intenta prioritzar l'ús del sistema *push*.

Google ofereix un sistema de notificacions *push* per dispositius Android, el servei *Google Cloud Messaging* (GCM). A la Fig. 41 es pot observar el funcionament del sistema:

1. El dispositiu mòbil es registra al servidor GCM (servidor de Google).
2. El dispositiu mòbil notifica al seu servidor propi que s'ha registrat al servidor GCM, enviant-li el seu *RegID*, que l'identifica al servidor GCM.
3. Quan el servidor propi vol enviar un missatge al dispositiu, envia el missatge al servidor GCM indicant-li el *RegID* del dispositiu.
4. El servidor GCM envia el missatge al dispositiu.

Per tant, només cal que el servidor guardi una taula que relacioni l'usuari amb el *RegID* i que enviï els missatges al servidor GCM que s'encarregarà de fer arribar el missatge al dispositiu corresponent, tal com s'esquematitza a la Fig. 41.

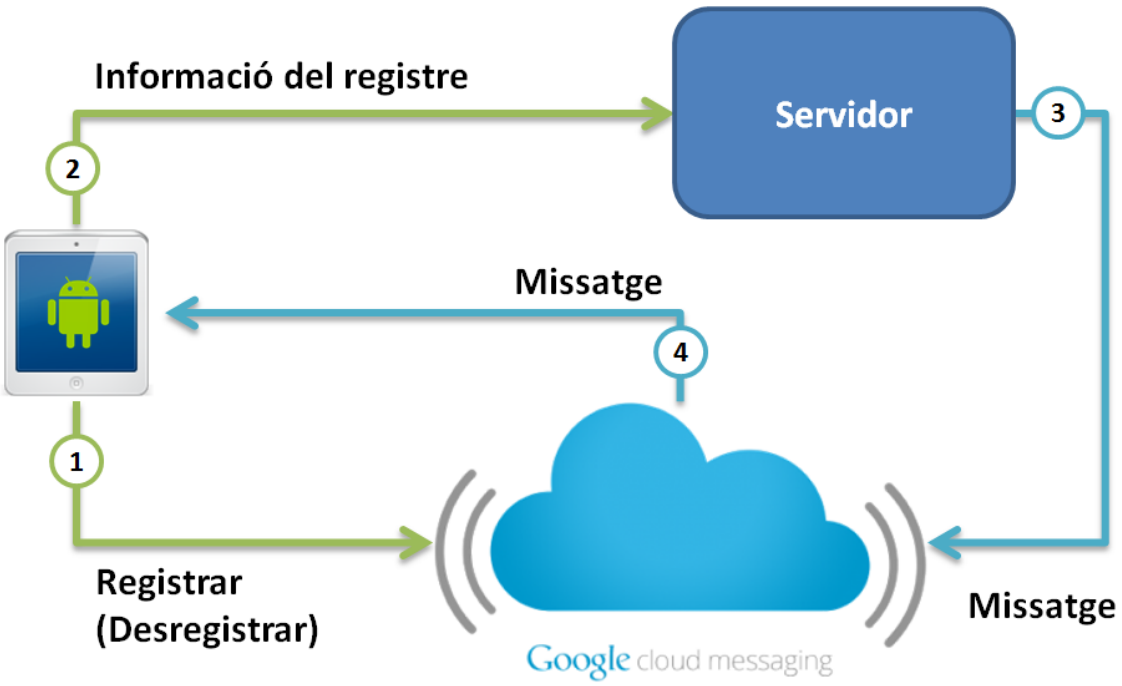

Fig. 41 Esquema de funcionament de Google Cloud Messaging.

Aquest sistema ha estat molt útil, entre d'altres coses, per poder notificar als *tablets* que cal que actualitzin el seu llistat d'aplicacions. Quan un metge assigna una aplicació a un beneficiari, el servidor envia una notificació *push* al dispositiu perquè actualitzi les dades. Sense notificacions *push*, el dispositiu hauria de comprovar cada cert temps si hi ha actualitzacions disponibles amb la conseqüent despesa de bateria i dades. Per a més informació sobre GCM es pot consultar [3].

B.2 Db4o

Db4o (veure [4]) és una base de dades orientada a objectes, la qual cosa significa que permet guardar objectes sense necessitat de mapar-los en taules relacionals. D'aquesta manera, s'abstrau al desenvolupador del disseny de les taules relacionals, de les *queries*, etc. A la Fig. 42 es mostra un exemple del codi.

```
ObjectContainer db = Db4oEmbedded.openFile(file);
db.store(app);
db.delete(app);
List<LauncherApp> l1 = db.query(LauncherApp.class);
```

Fig. 42 Exemple de línies de codi que interactuen amb la base de dades db4o.

La primera línia de la Fig. 42 defineix el fitxer on es guardarà la base de dades. Un cop fet això, ja es pot començar a utilitzar l'objecte *db*. La línia següent guarda l'objecte *app*, que és del tipus *LauncherApp*, a la base de dades. La següent comanda elimina aquest objecte. L'última línia llista tots els objectes de la base de dades de la classe *LauncherApp*.

En aquest projecte, aquesta base de dades és feta servir per l'aplicació Android per guardar el llistat d'aplicacions dels usuaris i informació sobre l'usuari.

B.3 Apache Maven

Maven és un software de gestió de projectes basat en el concepte *project object model* (POM). Permet fer multitud de tasques mitjançant els diferents connectors (*plug-ins*) que incorpora.

En aquest projecte es fa servir únicament per gestionar les llibreries. Enlloc de descarregar-les i agregar-les manualment al projecte, Maven permet fer-ho automàticament. Per a aconseguir-ho s'ha de configurar l'arxiu pom.xml (arxiu de configuració de Maven).

Cada llibreria s'anomena dependència, a [5] es pot buscar qualsevol llibreria i se n'obté la dependència que cal agregar a la configuració. Per exemple, buscant "gson" s'obtenen les dependències de les diferents versions de la llibreria, a la Fig. 43 es mostra la dependència de la versió 2.2.4.

```
<dependency>
        <groupId>com.google.code.gson</groupId>
        <artifactId>gson</artifactId>
        <version>2.2.4</version>
</dependency>
```

Fig. 43 Codi que caldria afegir a l'arxiu *pom.xml* per a agregar la llibreria GSON 2.2.4 al projecte.

Gestionar les llibreries mitjançant Maven dóna molta flexibilitat a l'hora de canviar de versió de llibreria. També és molt útil quan s'ha de compartir el projecte amb d'altres desenvolupadors perquè no hi ha arxius del sistema enllaçats amb el projecte (cosa que portaria problemes, ja que podrien no tenir els arxius a la mateixa ruta i llavors no es localitzarien correctament).

B.4 RoboGuice

RoboGuice és un *framework* per Android que permet simplificar el codi de l'aplicació (a continuació es demostrarà amb un exemple). Per altra banda, incorpora classes que donen més flexibilitat a l'hora de programar.

A la Fig. 44 i la Fig. 45 es mostra a través d'un exemple el sistema d'injecció de dependències que permet dur a terme *RoboGuice* i que simplifica significativament el codi. A la Fig. 44 es mostra un exemple del codi d'una *Activity*, en la qual es declaren un conjunt d'elements que després s'inicialitzen per poder ser utilitzats. Per exemple, l'element *name* és un *TextView*, és a dir, un component visual que mostra text. Primer, s'ha de declarar (línia 2), després s'ha de definir a quin element de l'arxiu XML de la vista es correspon (línia 11) i finalment ja es pot fer servir. En aquest cas, se li assigna un text perquè el mostri (línia 16).

```
1  class AndroidWay extends Activity {
2     TextView name;
3     ImageView thumbnail;
4     LocationManager loc;
5     Drawable icon;
6     String myName;
7
8     public void onCreate(Bundle savedInstanceState) {
9        super.onCreate(savedInstanceState);
10       setContentView(R.layout.main);
11       name      = (TextView) findViewById(R.id.name);
12       thumbnail = (ImageView) findViewById(R.id.thumbnail);
13       loc = (LocationManager)getSystemService(Activity.LOCATION_SERVICE);
14       icon      = getResources().getDrawable(R.drawable.icon);
15       myName    = getString(R.string.app_name);
16       name.setText( "Hello, " + myName );
17    }
18 }
```

Fig. 44 Exemple d'una A*ctivity* en la qual es pot veure la declaració de diversos elements i la inicialització dels mateixos a la funció *onCreate*.

El codi de la Fig. 45 fa exactament el mateix que el de la Fig. 44 però amb molt menys text. Això és gràcies a *RoboGuice*, que permet injectar els elements directament. Ara, el *TextView name* és injectat, és a dir, declarat i definit (línia 2) i ja pot ser emprat (línia 11).

```
1  class RoboWay extends RoboActivity {
2     @InjectView(R.id.name)              TextView name;
3     @InjectView(R.id.thumbnail)         ImageView thumbnail;
4     @InjectResource(R.drawable.icon)    Drawable icon;
5     @InjectResource(R.string.app_name)  String myName;
6     @Inject                             LocationManager loc;
7
8     public void onCreate(Bundle savedInstanceState) {
9        super.onCreate(savedInstanceState);
10       setContentView(R.layout.main);
11       name.setText( "Hello, " + myName );
12    }
13 }
```

Fig. 45 Codi de la Fig. 44 simplificat gràcies a l'ús de *RoboGuice.*

Per altra banda, una de les classes aportades per *RoboGuice* que ha estat molt útil per al projecte és la *SafeAsyncTask*. A Android, hi ha certes tasques que no es poden dur a terme al *thread* principal, de manera que cal crear una tasca asíncrona que les realitzi, implementant per a això la classe *AsynkTask*. Tanmateix, aquesta classe no permet gestionar de manera senzilla els errors, mentre que la classe *SafeAsyncTask* incorporada per *RoboGuice* la simplifica i permet fer una gestió més senzilla dels errors.

Al projecte, totes les tasques asíncrones són instàncies de la classe *SafeAsyncTask,* i es fa servir la injecció de dependències a totes les *Activities*.

B.5 Spring MVC

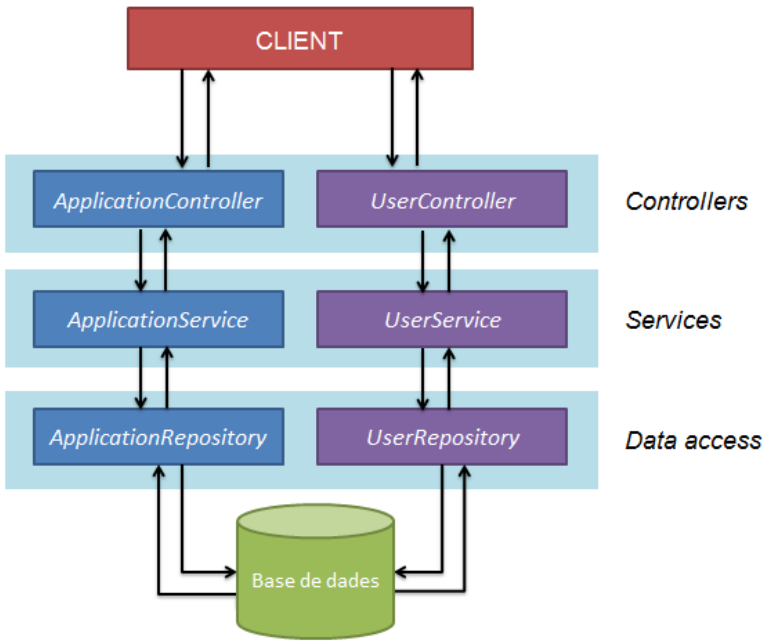

Fig. 46 Exemple d'arquitectura de Spring MVC.

El servidor ha estat desenvolupat amb el *framework Spring MVC*. L'arquitectura general de *Spring MVC* és la que es pot observar a Fig. 46. El client ataca un *Controller*, aquest al seu *Service* i aquest al seu *Repository,* que és l'encarregat de l'accés a la base de dades.

Un *Controller* està format per diferents funcions a cada una de les quals se li pot assignar una URL diferent. Per exemple, el *Controller* d'aplicacions podria tenir una funció *getUserApps* a la que es pogués accedir a través de la URL «[adreça del servidor]/apps/listApps».

Al projecte, totes les connexions des dels dispositius mòbils es fan contra el *MobileController* (excepte les relacionades amb trucades, tal com s'ha explicat a l'apartat 4.2) que proporciona tots els serveis als dispositius mòbils. El *MobileControler* fa peticions *ApplicationService*, *AuthService, PushService*, etc. No es fan servir directament els *Controllers* d'aplicacions, d'autenticació, etc., perquè totes les peticions mòbils s'autentiquen d'una manera concreta i és més fàcil de gestionar si ataquen sempre a un mateix *Controller*.

B.6 GSON

GSON és una llibreria de Google que permet convertir objectes Java al llenguatge JSON i viceversa. A continuació es mostra un petit exemple de com funciona el llenguatge JSON. Es vol convertir un objecte del tipus persona, que té un nom, un cognom, una edat, i una feina, on la feina conté el nom de la companyia, el salari i la posició dins l'empresa, tal com es mostra a la Fig. 47.

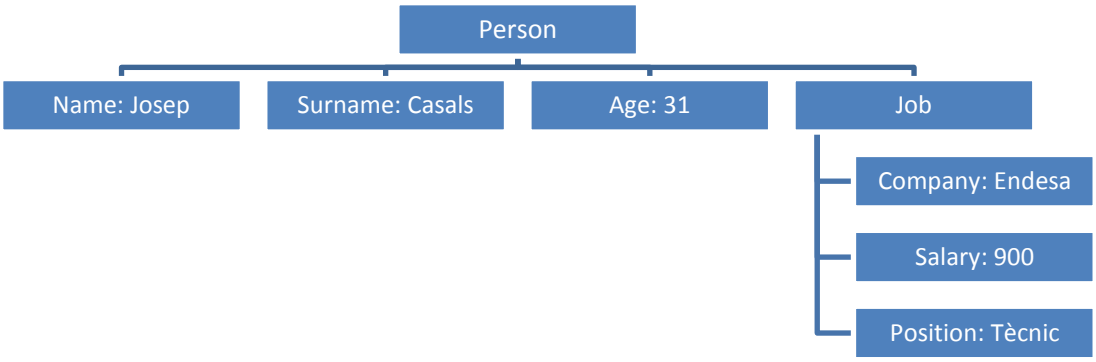

Fig. 47 Exemple d'objecte del tipus persona: conté les seves dades i a dins un objecte del tipus *job* que descriu la seva feina.

GSON convertirà aquest objecte al llenguatge JSON, que consisteix en una cadena de caràcters amb l'estructura que es pot observar a la Fig. 48.

```
{
        "name":"Josep",
        "surname":"Casals",
        "age":31,
        "job":{
                "company":"Endesa",
                "position":"Tècnic",
                "salary":900
        }

}
```

Fig. 48 Objecte de la Fig. 47 en llenguatge JSON.

Després, permetrà obtenir un altre cop l'objecte Java sense afegir gaire temps de processat. En resum, aquesta llibreria permet enviar objectes Java complexos sense necessitat d'introduir tan temps de processat com introduirien altres mètodes com, per exemple, XML. Al projecte es fa servir GSON per a serialitzar objectes en totes les comunicacions client-servidor en les quals cal enviar objectes complexos: el llistat d'aplicacions, les dades de l'usuari, etc.

ANNEX C AMPLIACIÓ DEL DISSENY VISUAL DEL LAUNCHER

En aquest annex s'explica com el Launcher suporta, canvis d'orientació, d'idioma i com s'aconsegueix que sigui compatible amb dispositius amb pantalles de diferents mides.

C.1 Canvis d'orientació

El Launcher està preparat per suportar canvis d'orientació del dispositiu, és a dir, les vistes s'adapten a l'orientació vertical o horitzontal segons correspongui. Això es fa de manera automàtica en alguns casos, com per exemple a la *Vista d'Aplicacions Instal·lades al Dispositiu*, que s'adapta automàticament quan hi ha un canvi d'orientació. En d'altres casos s'han de crear dues vistes diferents: una per al dispositiu en horitzontal i una per al dispositiu en vertical, que és el cas de la *Vista Principal* (veure Fig. 49).

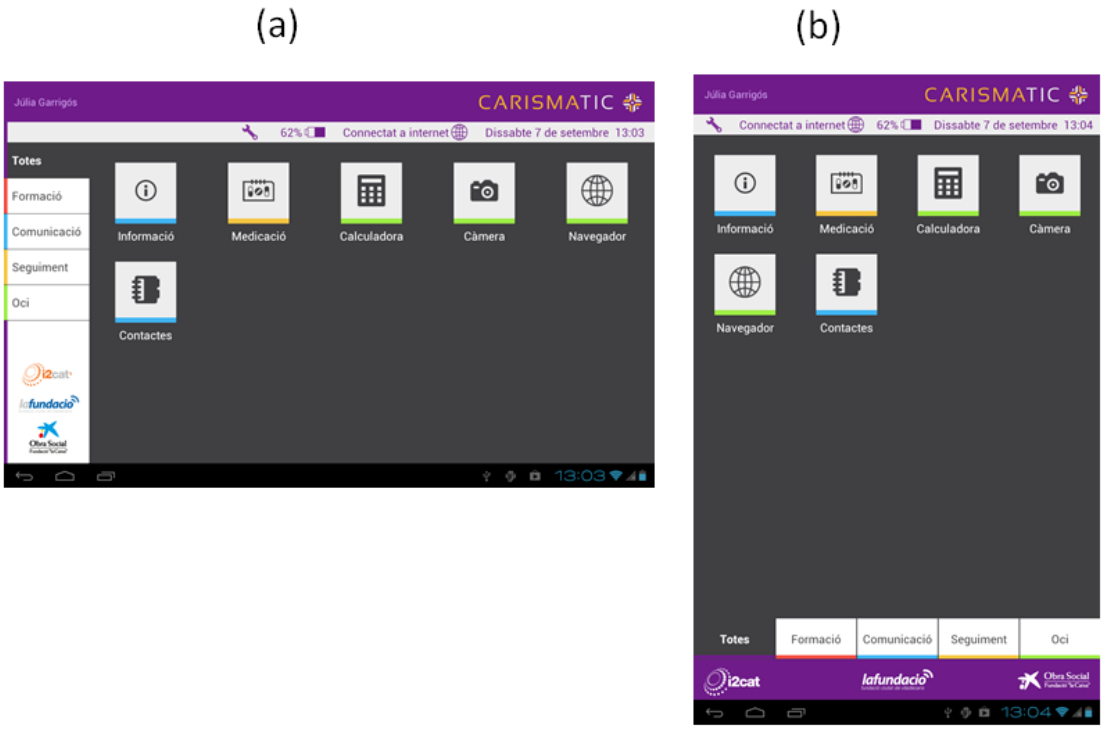

Fig. 49 *Vista Principal* en orientació horitzontal (a) i vertical (b).

Aquests canvis impliquen que s'ha de crear una carpeta per les vistes (*layouts*) en horitzontal (*landscape*) i una altra per les vistes en vertical (*portrait*). Android està preparat per triar la vista corresponent a l'orientació. Perquè ho faci s'han de crear les carpetes *layout-land* (per les vistes horitzontals) i *layout-port* (per les vistes verticals).

C.2 Compatibilitat amb diferents mides de pantalla

Tal com es demana als requeriments, el Launcher és compatible amb *tablets* amb pantalles de 7'' i 10''. En alguns casos s'ha de crear una vista per cada mida de pantalla. Perquè el sistema operatiu sàpiga quina és la vista que ha de mostrar s'ha de crear una carpeta de vistes (*layouts*) per a cada mida.

A la Fig. 50 es mostra el nom que dóna Android a les diferents mides de pantalla. Tal com es pot veure, fa falta la carpeta *layout-large* per les vistes dels dispositius de 7 polzades i la carpeta *layout-xlarge* per les vistes dels dispositius de 10 polzades. Si alguna vista és comuna per ambdues mides es podrà posar a la carpeta genèrica *layout*. S'ha establert, per motius d'usabilitat, que els dispositius de 7'' empraran sempre vistes en orientació vertical.

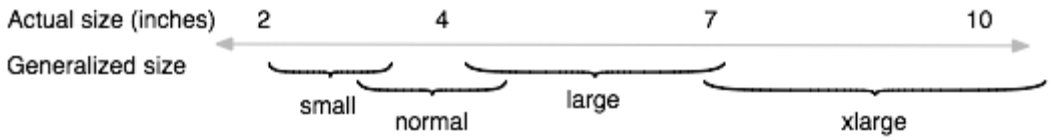

Fig. 50 Denominació del tipus de pantalla en funció de les seves polzades. Font: [2].

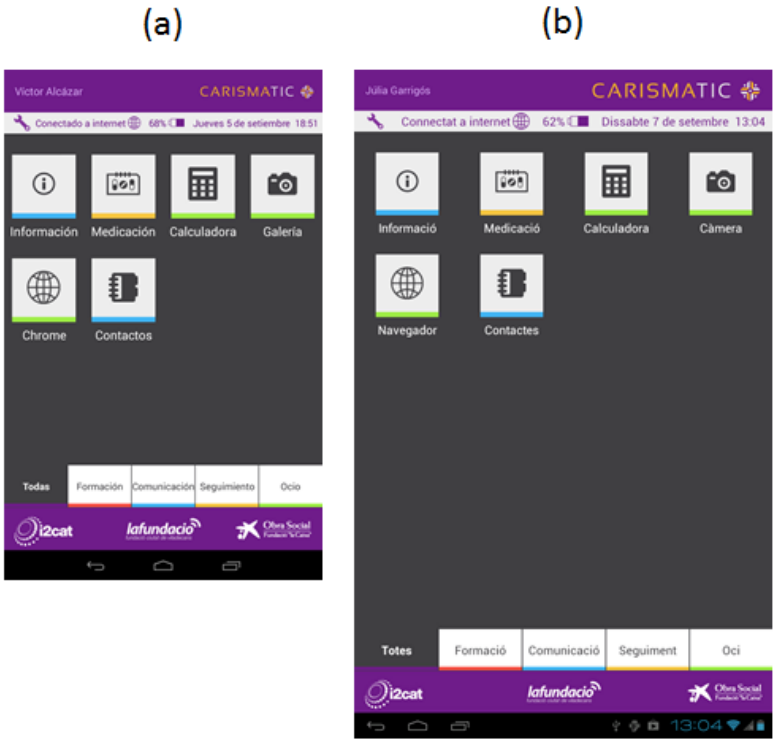

Fig. 51 *Vista Principal* en un dispositiu de 7'' (a) i en un de 10'' en orientació vertical (b).

Si es té en compte que cada una de les vistes pot ser per a les orientacions horitzontal i vertical resulta que fan falta les carpetes següents:

- *layout-xlarge-land*: vistes per a 10 polzades i orientació horitzontal
- *layout-xlarge-port:* vistes per a 10 polzades i orientació vertical
- *layout-large:* vistes per a 7 polzades (totes les quals, tal com s'ha dit, seran verticals)
- *layout-xlarge*: vistes per a 10 polzades independents de l'orientació
- *layout*: vistes independents de la mida de la pantalla i de l'orientació.

A la Fig. 51 es pot veure la *Vista Principal* en les dues mides de pantalla suportades: 7'' i 10'' (en el segon cas, en orientació vertical, per poder comparar amb el dispositiu de 7'').

C.3 Multi-idioma

L'aplicació ha d'estar disponible en català i en castellà i a més ha de permetre, de forma fàcil, afegir un tercer idioma. L'aplicació triarà l'idioma en què estigui configurat el dispositiu mitjançant un sistema proporcionat per Android que consisteix en crear una carpeta per a cada idioma i posar dins els arxius que contenen les cadenes de caràcters (*strings*) que fan falta per a l'aplicació. El Launcher té una carpeta *values-es* que conté l'arxiu *Strings.xml* amb els textos en castellà i una carpeta *values-ca* que conté els textos en català.

Fins ara la majoria de captures que s'han vist estaven en català; a la Fig. 52 es pot veure la *Vista Principal* en castellà.

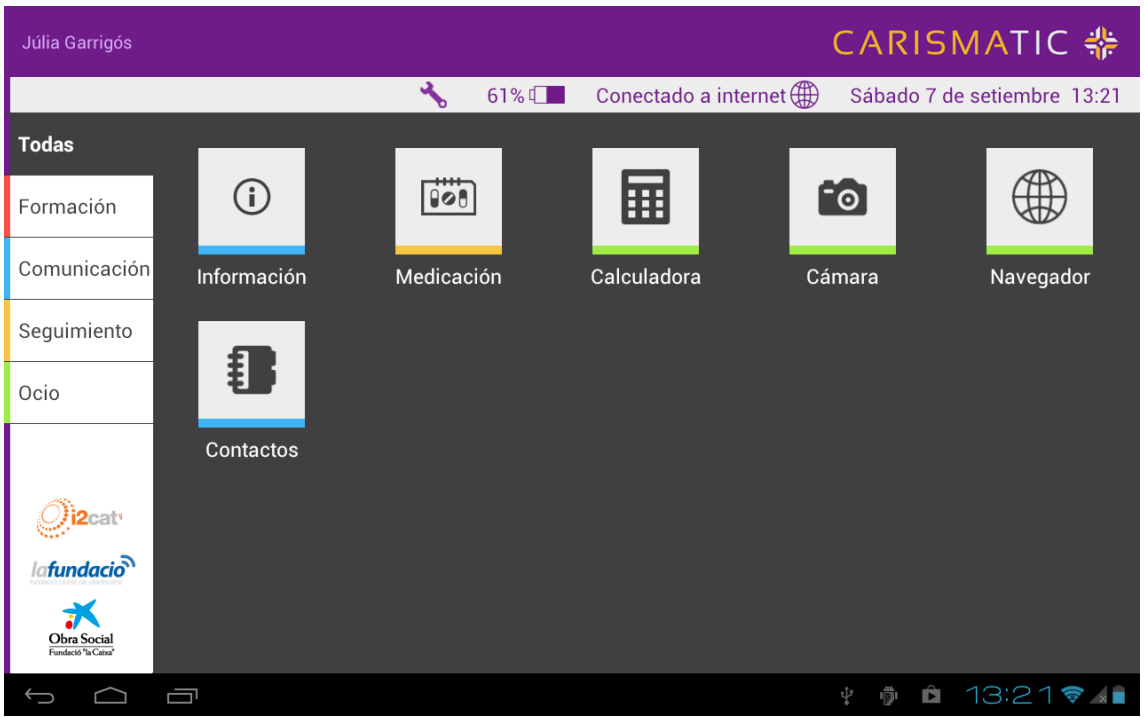

Fig. 52 Captura de la *Vista Principal* en castellà.